

TEC US 2011: PowerShell Deep Dive: Bruce Payette – Why Does it Work that Way? Inside the PowerShell Runtime

These are the scripts Bruce was using. You can also find his:

- Slides & video recording here: <http://dmitrysotnikov.wordpress.com/2011/08/30/video-bruce-payette-inside-powershell-runtime/>

Main files

ClosureExample.ps1

```
#
# Object === data + code
#
# Closure === code + data
#

#
# Function factory - returns closures that scale their argument
# by the the value of $x
#
function scale ($x)
{
    $sb = {param($y) $x * $y}
    $sb.GetNewClosure()
}

#
# Define a scale-by-5 scriptblock...
#
$by5 = scale 5
1..10 | foreach { & $by5 }

#
# Define an actual function
#
$function:by5 = scale 5

#
# And use it ...
by 3

#
# Fancier function - uses "advanced function" metadata...
#

function New-ScaleFunction
```

```

{
    param (
        [parameter(Mandatory=$true)]
        [string]
            $functionName,
        [parameter(Mandatory=$true)]
        [int]
            $scaleFactor
    )

    # "Advanced" scriptblock
    $sb = {
        param (
            # Can read value from pipeline as well as command line...
            [parameter(ValueFromPipeline=$true,Mandatory=$true)]
            [int]
                $valueToScale
        )

        process { $valueToScale * $scaleFactor }
    }

    # Define a scale function at the global scope...
    Set-Item "function:global:$functionName" $sb.GetNewClosure()
}

```

```
New-ScaleFunction by4 4
```

```
by4 10
```

```
1..10 | by4
```

demo.ps1

```

# PowerShell is an Expression-oriented language
# - Everything returns a value

```

```

# For cmd.exe - this is not surprising
cmd /c "dir & ipconfig & ver"

```

```

# Nor is this...
Get-Date; Dir; Get-WmiObject Win32_BIOS

```

```
# Is this surprising? Probably not...
```

```
function foo
```

```

{
    1
    2
    3
}

```

```
# But this probably is...
```

```
function test-arraylist ()
```

```
{
```

```

    $al = New-Object System.Collections.ArrayList
    $al.Add(1)
    $al.Add(2)
    $al.Add(3)
}

#####
#
# Assignment works in statements
# - can assign directly into a variable in V2...
#
$x = foreach ($i in 1..10) { if ( $i % 2) { $i } }

# Can stream a statement into a loop...
$( foreach ($i in 1..10) { if ( $i % 2) { $i } } ) | foreach { "Item is $_" }

#####
# Conversions, collections and $null

[bool] 1
[bool] 0
[bool] "True"
[bool] "true"
[bool] $null #<-- false
[bool] @($null) #<--- false
[bool] @($null, $null) #<--- true!!!

# Collections of null - pathological consistency...

foreach ($v in $null, $null, $null) { "ping" } #<-- loop 3 times
foreach ($v in $null, $null) { "ping" } #<-- loop 2
foreach ($v in $null) { "ping" } #<-- loop 1

#####
#
# Steppable pipelines
#
$sb = { sort -descending | select -first 3 }
$sp = $sb.GetSteppablePipeline()
$sp.Begin.OverloadDefinitions
$sp.Begin($true)
$sp.Process(5)
$sp.Process(3)
$sp.Process(10)
$sp.End()

# Remoting...
$s = nsn
$spid ; $spid = icm $s { $pid } ; $spid

Get-Process -id $spid

5,4,2,6,1,2,7 | icm $s { $input | sort -Descending }

5,4,2,6,1,2,7 | icm $s { sort -Descending ; "Hi" }

```

```

$P =
[System.Management.Automation.PowerShell]::Create().AddCommand("sort").AddParameter("descending")
$P.Invoke((4,5,2,6))

icm 5,4,2,6,1,2,7 | icm { sort -Descending ; "Hi" }

$P = [PowerShell]::Create().AddScript('$input | sort -Descending ; "Hi"')
$P.Invoke((4,5,2,6))

$P = {sort -descending | select -first 3 }.

$S = nsn
$result = icm $S { @{a=12.3; sb = { "123" } } }

$R = $S.Runspace
$P = [powershell]::Create().AddCommand('write-output')
$P.runspace = $R

$P.Invoke( , (@{ a = {123} } ) )

```

dynamic_modules_example.ps1

```

# module templates - scriptblocks used to initialize the command-set module instances

$ct1 = {
    function foo { "Foo1" }
    function bar { "Bar1" }
    function baz { "Baz1" }
}

$ct2 = {
    function foo { "Foo2" }
    function bar { "Bar2" }
}

# command-set module instances - contain domain-specific commands + global
# currently singletons but could use multiple instances

$csm1 = new-module -Function @() $ct1
& $csm1 Set-Variable CommandSetName "Command Set 1"

$csm2 = new-module -Function @() $ct2
& $csm2 Set-Variable CommandSetName "Command Set 2"

#
# Domain executors - global since the nesting order is
# arbitrary (either one can call the other or itself...)
# but could be defined as module local
#
function global:Using-Cs1 ($cmd)
{

```

```

    & $csm1 $cmd
}

function global:Using-Cs2 ($cmd)
{
    & $csm2 $cmd
}

#
# Example application..
#

Using-Cs1 {
    Write-Output "Top-level: in $CommandSetName"
    Foo #<- this will run the foo in command set 2
    bar
    baz
    $CS1value = 1
    "Initial CS1value = $CS1value"
    Using-Cs2 {
        Write-Output "Nested 1: in $CommandSetName"
        Foo #<- this will run the foo from command set 2
        bar
        baz #<- error - this is not defined in Command set 2
        Using-Cs1 {
            Write-Output "Nested 2: in $CommandSetName"
            foo
            bar
            $CS1value++ #<- update the variable - this
            "Updated CS1value = $CS1value"
            Using-Cs2 {
                Write-Output "Nested 1: in $CommandSetName"
                foo
            }
            Write-Output "Nested 2: in $CommandSetName"
            baz
        }
    }
    "Final CS1value = $CS1value - not modified because it is a local
    variable..."
}

function new-myobj { param ([int] $i, [hashtable] $h) New-Object psubject -
    property $psboundparameters }

```

generateFib.ps1

```

#
# Reactive fib generator - emits infinite sequence...
#

function Generate()
{
    ($n1 = 1)
    ($n2 = 1)
    while($true)

```

```

    {
        $n3 = $n1 + $n2
        $n1 = $n2
        $n2 = $n3
        $n3
    }
}

#
#
generate | foreach {
    $_
    if ((Read-Host "Enter for next number, q to quit") -eq "q")
        { break }}

```

HistoryGUI

historygui.psm1

```

try
{
    Import-Module $PSScriptRoot/xamltools.psm1
}
catch
{
    throw $_.Message
}

#####
# History tab

function updateHistory
{
    $hlist.Items.Clear()
    $commands = @(Get-History -Count ([int16]::MaxValue) |
        where {$_ -match $script:patternControl.Text})
    if ($commands)
    {
        [array]::reverse($commands)
        $commands | % { [void] $hlist.Items.Add($_) }
    }
}

function returnHistory
{
    $script:result = ( $hlist.SelectedItem |
        foreach { $_.CommandLine } ) -join ';'
    $f.Close()
}

function setupHistory
{
    $script:hlist = $f.FindName("HistoryListBox")
    $script:patternControl = $f.FindName("Pattern")
}

```

```

    $okButton = $f.FindName("okButton")
    $patternControl.Text = $pattern
    $patternControl.add_TextChanged({ updateHistory })
    $hlist.add_MouseDoubleClick({ returnHistory })
    $okButton.add_Click({ returnHistory })
    $script:result = $null
    updateHistory
}

#####
# directory list tab

function updateDirList
{
    $dlist.Items.Clear()
    dirs | where { $_ -match $dPatternControl.Text } |
        foreach { [void] $dlist.Items.Add($_) }
}

function setupDirList
{
    $script:dlist= $f.FindName("DirectoryListBox")
    $dlist.add_MouseDoubleClick({
        $target = $dlist.SelectedItem -replace '^ *[0-9]+ '
        $host.UI.WriteVerboseLine("cd'ing to '$target'")
        cd $target
        $f.Close()
    })
    $runDirectory = $f.FindName("runDirectory")
    $runDirectory.add_Click({
        $target = $dlist.SelectedItem -replace '^ *[0-9]+ '
        $host.UI.WriteVerboseLine("cd'ing to '$target'")
        cd $target
        $f.Title = "PowerShell Command History: $pwd"
    })
    $okDirectory = $f.FindName("okDirectory")
    $okDirectory.add_Click({
        $script:result = $dlist.SelectedItem -replace '^ *[0-9]+ ', 'cd'
        $f.Close()
    })
    $script:dPatternControl = $f.FindName("DirectoryPattern")
    $dPatternControl.add_TextChanged({ updateDirList })
    updateDirList
}

#####
# snippet tab

function updateSnippets
{
    $sList.items.Clear()
    Get-Content "$PSScriptRoot/snippets.ps1" |
        where { $_ -match $sPatternControl.Text } |
        foreach { [void] $slist.Items.Add($_) }
}

```

```

function setupSnippets
{
    $script:slist= $f.FindName("SnippetListBox")
    $script:sPatternControl = $f.FindName("SnippetPattern")
    $okSnippet = $f.FindName("okSnippet")
    $okSnippet.add_Click({
        $script:result = $slist.SelectedItem
        $f.Close()
    })
    $sPatternControl.add_TextChanged({ updateSnippets })
    $hlist.add_MouseDoubleClick({
        $script:result = $slist.SelectedItem
        $f.Close()
    })
    updateSnippets
}

function Invoke-GuiHistory
{
    param ($pattern)

    $xamlPath = Join-Path $PSScriptRoot historygui.xaml
    $script:f = Invoke-Xaml $xamlPath
    $f.Title = "PowerShell Command History: $pwd"

    setupHistory
    setupDirList
    setupSnippets

    [void] $f.ShowDialog()

    if ($host.Name -match "ise")
    {
        $psise.CurrentPowerShellTab.CommandPane.Text = $script:result
    }
    else
    {
        if ($script:result)
        {
            $host.UI.WriteVerboseLine("Selection is in clipboard")
            [System.Windows.Clipboard]::SetText($script:result)
        }
    }
}

#
# If loaded from the ISE, set up an add-on menu entry
#
if ($host.Name -match "powershell ise" -and
    -not ($psise.CurrentPowerShellTab.AddOnsMenu.Submenus |
        where {$_.DisplayName -match 'History browser'}))
{
    $psISE.CurrentPowerShellTab.AddOnsMenu.Submenus.Add(
        "History Browser!", {Invoke-GuiHistory}, "Alt+h")
}

Set-Alias igh Invoke-GuiHistory

```

`Export-ModuleMember -Function Invoke-GuiHistory -Alias igh`

historygui.xaml

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="PowerShell Command History" Height="340" Width="511"
MinWidth="300" MinHeight="200" WindowStartupLocation="CenterScreen">

<TabControl>

    <TabItem>
        <TabItem.Header>
            <TextBlock>Command History</TextBlock>
        </TabItem.Header>

<!-- History tab -->

        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition /> <RowDefinition Height="50"/>
            </Grid.RowDefinitions>
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/> <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/> <RowDefinition />
                </Grid.RowDefinitions>
                <Label Content="Command" Margin="5, 2, 10, 0" FontSize="12" />
                <TextBox Grid.Column="1" Height="25" Margin="5, 2, 5, 2"
                    Name="Pattern" />
                <ListBox Grid.ColumnSpan="2" Grid.Row="1" Name="HistoryListBox"
                    HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
                    FontFamily="Consolas"
                    Margin="5, 2, 5, 5" >
                    <ListBox.ContextMenu>
                        <ContextMenu>
                            <MenuItem Header="_Bold" IsCheckable="True"/>
                            <MenuItem Header="_Italic" IsCheckable="True"/>
                            <Separator />
                            <MenuItem Header="I_ncrease Font Size" />
                            <MenuItem Header="_Decrease Font Size" />
                        </ContextMenu>
                    </ListBox.ContextMenu>
                </ListBox>

            </Grid>

        </Grid>

    </TabItem>

</TabControl>
```

```

<Grid Grid.Row="1" HorizontalAlignment="Right"
Margin="0, 10, 10, 10">
  <Grid.ColumnDefinitions>
    <ColumnDefinition /> <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button x:Name="okButton" IsDefault="True" Grid.Column="0"
Content="OK" Height="25" Width="80" Margin="0, 2, 10, 2" />
  <Button IsCancel="True" Grid.Column="1"
Content="Cancel" Height="25" Width="80" Margin="0,2,10,2" />
</Grid>
</Grid>

```

```

</TabItem>

```

```

<!-- Snippet tab -->

```

```

<TabItem>
<TabItem.Header>
  <TextBlock>Snippets</TextBlock>
</TabItem.Header>

```

```

<Grid>
<Grid.RowDefinitions>
  <RowDefinition /> <RowDefinition Height="50"/>
</Grid.RowDefinitions>
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/> <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/> <RowDefinition />
  </Grid.RowDefinitions>
  <Label Content="Snippets" Margin="5, 2, 10, 0" FontSize="12" />
  <TextBox Grid.Column="1" Height="25" Margin="5, 2, 5, 2"
Name="SnippetPattern" />
  <ListBox Grid.ColumnSpan="2" Grid.Row="1" Name="SnippetListBox"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
FontFamily="Consolas"
Margin="5, 2, 5, 5" >
<ListBox.ContextMenu>
  <ContextMenu>
    <MenuItem Header="_ Bold" IsCheckable="True"/>
    <MenuItem Header="_ Italic" IsCheckable="True"/>
    <Separator />
    <MenuItem Header="I_ncrease Font Size"/>
    <MenuItem Header="_ Decrease Font Size"/>
  </ContextMenu>
</ListBox.ContextMenu>

```

```

</ListBox>
</Grid>
<Grid Grid.Row="1" HorizontalAlignment="Right"
Margin="0, 10, 10, 10">
  <Grid.ColumnDefinitions>
    <ColumnDefinition /> <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button x:Name="okSnippet" IsDefault="True" Grid.Column="0"
Content="OK" Height="25" Width="80" Margin="0, 2, 10, 2" />
  <Button IsCancel="True" Grid.Column="1"
Content="Cancel" Height="25" Width="80" Margin="0,2,10,2" />
</Grid>
</Grid>

```

```

</TabItem>

```

```

<!-- Directory tab -->

```

```

<TabItem>
  <TabItem.Header>
    <TextBlock>Directory</TextBlock>
  </TabItem.Header>

  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition /> <RowDefinition Height="50"/>
    </Grid.RowDefinitions>
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/> <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/> <RowDefinition />
      </Grid.RowDefinitions>
      <Label Content="Directory" Margin="5, 2, 10, 0" FontSize="12" />
      <TextBox Grid.Column="1" Height="25" Margin="5, 2, 5, 2"
Name="DirectoryPattern" />
      <ListBox Grid.ColumnSpan="2" Grid.Row="1" Name="DirectoryListBox"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
FontFamily="Consolas"
SelectionMode="Multiple" Margin="5, 2, 5, 5" >
    <ListBox.ContextMenu>
      <ContextMenu>
        <MenuItem Header="_Bold" IsCheckable="True" />
        <MenuItem Header="_Italic" IsCheckable="True" />
        <Separator />
        <MenuItem Header="I_ncrease Font Size" />
      </ContextMenu>
    </ListBox.ContextMenu>

```

```

    <MenuItem Header="_Decrease Font Size" />
</ContextMenu>
</ListBox.ContextMenu>
</ListBox>

</Grid>
<Grid Grid.Row="1" HorizontalAlignment="Right"
Margin="0, 10, 10, 10">
  <Grid.ColumnDefinitions>
    <ColumnDefinition /> <ColumnDefinition /> <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button x:Name="runDirectory" Grid.Column="0"
Content="Run" Height="25" Width="80" Margin="0, 2, 10, 2" />
  <Button x:Name="okDirectory" IsDefault="True" Grid.Column="1"
Content="OK" Height="25" Width="80" Margin="0, 2, 10, 2" />
  <Button IsCancel="True" Grid.Column="2"
Content="Cancel" Height="25" Width="80" Margin="0,2,10,2" />
</Grid>

</Grid>
</TabItem>
</TabControl>
</Window>

```

snippets.ps1

```

#####
#
# String operations
#
$a = 2; "a is $a" # variable expansion in double-strings
"The date is $(Get-Date)" # subexpression evaluation in double-
quoted strings
"a`t tab and a `n" # escape sequence for tab and newline,
escape character is ``
'a is $a' # no expansion in single-quoted strings
"hello" + "there" # string concatenation
"hello" -like "he*" # string match wildcard
"hello" -match "el" # string match regex
"hello" -replace 'ell.*$', "i there" # string replace regex
"abc" * 10 # string multiplication
"a, b, c" -split ', *' # split string using regex
-split "a b c" # unary split on whitespace
"abcdefg"[0] # indexing, origin 0
"abcdefg"[1..3] # indexing with range, returns char
array
"abcdefg".Substring(3) # substring from position to end of
string
"abcdefg".Substring(2,3) # substring from position, n characters

#####
#
# Array operations

```

```

#
$a = 1,2,3,4           # literal array using comma operator,
no brackets required
$a = ,1                # literal array of one element with
unary comma operator
$a = @()               # empty array
(1,2,3,4,5).Length    # length of an array
$a = @( dir )          # force result of command to be an
array
$a = 1,2,3,4 + 5,6     # array concatenation
$a[0]                  # array index, origin zero
$a[2..3]               # array range
"one", "two", "three" -match '[nw]' # array regex match to extract elements
"one", "two", "three" -like 't*'    # array wildcard match to extract
elements
@(1,2,3,4,5,6,7 -gt 2) -lt 4        # extract elements from array using
comparison operations
-join "a","b","c"                   # unary join without spaces
"a","b","c" -join "+"               #

```

```
#####
```

```

#
# Conditional statements
#
# statement conditional fi else endif

```

```

if ( $x -gt 10 )
{
    # ...
}
elseif ($x -gt 100) # elseif is a keyword
{
    # ...
}
else
{
    # ...
}

```

```
#####
```

```

#
# looping statements
#
# statement looping while
while ( $x -lt 10)
{
    # ...
}

# statement looping for
for ($i = 1; $i -lt 10; $i++)
{
    $i * 2
}
#statement looping foreach

```

```

foreach ($i in Get-Process)
{
    # ...
}

#####
#
# Defining functions
#

# statement simple function
function foo ($x)
{
    $x * 2
}

# statement simple function using param statement
function foo
{
    param ($x)
    $x * 2
}

# statement begin/process/end function
function foo ($x)
{
    begin
    {
        # ...
    }
    process
    {
        # ...
    }
    end
    {
        # ...
    }
}

# statement advanced function
function foo
{
    [CmdletBinding(
        DefaultParameterSet="parametersetname",
        ConfirmImpact=$true,
        SupportsShouldProcess=$true,
    )]
    param (
        [Parameter(Mandatory=$true,
            Position=0,
            ParameterSetName="set1",
            ValueFromPipeline=$false,
            ValueFromPipelineByPropertyName=$true,
            ValueFromRemainingArguments=$false,
            HelpMessage="some help this is")]
        [int]

```

```

        $p1 = 0
    )
begin
{
    # ...
}
process
{
    # ...
}
end
{
    # ...
}
}

#####
#
# The format operator
#
"{0}+{1} is {2}, today is {3}" -f 1,1,2,(Get-Date).dayofweek
# Format specifiers
#{0}      Display a particular element.
"{0} {1}" -f "a","b"      # -> a b
#{0:x}    Display a number in Hexadecimal.
"0x{0:x}" -f 181342      # -> 0x2c45e
#{0:X}    Display in hex, letters in uppercase
"0x{0:X}" -f 181342      # -> 0x2C45E
#{0:dn}   Display number left-justified, padded with zeros
"{0:d8}" -f 3            # -> 00000003
#{0:p}    Display a number as a percentage.
"{0:p}" -f .123 12.30    # -> %
#{0:C}    Display a number as currency
"{0:c}" -f 12.34         # -> $12.34
#{0,n}    Display with field width n, left aligned.
"|{0,5}|" -f "hi"        # -> | hi|
#{0,-n}   Display with field width n, right aligned.
"\"|{0,-5}|" -f "hi"      # -> |hi |
#{0:hh} {0:mm} Displays the hours and minutes from a DateTime value.
"{0:hh}:{0:mm}" -f (Get-Date) # -> 01:34
#{0:C}    Display using the currency symbol for the current culture.
"\"|{0,10:C}|" -f 12.4    # -> | $12.40|

#####
#
# recursive directory search - file a file name
dir -recurse -filter *name*
dir -recurse -filter *name* c:\users\
#
# directory search, show directories only
dir -recurse | where {$.PSIsContainer}
# directory search, show files only
dir -recurse | where { -not $.PSIsContainer}
# directory search, recursive, looking for matching strings in text files
dir -recurse -filter *.txt | Select-String 'text.*to.*match' # regex match
dir -recurse -filter *.txt | Select-String -wildcard '*text.*to.*match*' #
wildcard match

```

xamltools.psm1

```
$mode = [System.Threading.Thread]::CurrentThread.ApartmentState
if ($mode -ne "STA")
{
    throw "This script can only be run when powershell is started with -sta"
}

Add-Type -AssemblyName PresentationCore, PresentationFramework

function Invoke-Xaml
{
    param(
        [Parameter(Mandatory=$true)] $path,
        [switch] $show
    )

    $stream = [System.IO.StreamReader] (resolve-path $Path).ProviderPath
    $form = [System.Windows.Markup.XamlReader]::Load($stream.BaseStream)
    $stream.Close()

    if ($show)
    {
        $form.ShowDialog()
    }
    else
    {
        $form
    }
}
```

WebBrowser

basicWebformExample.ps1

```
#
# A simple
#

# Must be running STA to use the browser control

$mode = [System.Threading.Thread]::CurrentThread.ApartmentState
if ($mode -ne "STA")
{
    throw "This script can only be run when powershell is started with -sta"
}

#
# Uses the Form function from WPIAforms
# module to build the basic frame for the UI
```

```

#
Import-Module ./wpiaforms

#
# HTML defining the form
#
$pageText = @"
<html>
<head>
    <title>
        PowerShell Commands
    </title>
</head>
<body bgcolor="lightblue">
    <a href="http://microsoft.com">Go to microsoft!</a>
    <ul>
        <H2 align="center">PowerShell Commands Page</H2>
    </ul>
    <Table cellpadding="2" cellspacing="4" border="0">
        <tr>
            <td>
                <button id="button1">List Processes</button>
            </td>
            <td>
                <button id="button2">List Services</button>
            </td>
            <td>
                <button id="button3">List Files</button>
            </td>
        </tr>
    </Table>
    <p>
        <textarea name="OutputArea" rows=10 style="width: 100%"></textarea>
    </p>
    <p>
        <select size="5" name="AvailableComputers" style="width:100%"
></select>
    </p>
</body>
</html>
"@

#
# Function to bind event handlers to controls
#
function global:Add-EventHandler
{
    [cmdletbinding()]
    param(
        [parameter(mandatory=$true)]
            $Document,
        [parameter(mandatory=$true)]
            $ControlId,
        [parameter(mandatory=$true)]
            $EventName,
        [parameter(mandatory=$true)]
            [scriptblock]

```

```

        $Handler
    )

    $control = $Document.Document.GetElementById($ControlId)
    $control.AttachEventHandler($EventName, $handler)
}

#
# Function to load and invoke an HTML form
#
function Invoke-HtmlApplication
{
    param (
        $pageText,
        $initializationScript
    )

    # Build the basic form object
    $global:f = form form @{
        text = "Browser"
        size = point 800 600
        controls = {
            ($global:w = form webbrowser @{
                dock = "fill"
                DocumentText = $pageText
            })
        }
    }

    # Load the HTML into the form
    $w.add_DocumentCompleted($initializationScript)
    # Show the form...
    $f.ShowDialog()
}

#
# The actual application: passes the HTML to define the form
# and the scriptblock to bind the event handlers...
#
Invoke-HtmlApplication $pageText {

    Add-EventHandler $w button1 onclick {
        $text = $w.Document.GetElementById("OutputArea")
        $text.InnerText = Get-Process | ft -auto | out-string
    }

    Add-EventHandler $w button2 onclick {
        $text = $w.Document.GetElementById("OutputArea")
        $text.InnerText = Get-Service | ft -auto | out-string
    }

    Add-EventHandler $w button3 onclick {
        $text = $w.Document.GetElementById("OutputArea")
        $text.InnerText = dir ~/documents | ft -auto | out-string
    }

    $AvailableComputers = $w.Document.GetElementById("AvailableComputers")
    foreach ($i in 1..20)

```

```

{
    $objOption = $AvailableComputers.Document.createElement("OPTION")
    $objOption.InnerText = "Option No. $i"
    $objOption.SetAttribute("Value", $i)
    if ($i -eq 2 -or $i -eq 4)
    {
        # $objOption.style.backgroundColor = "yellow"
    }
    $AvailableComputers.AppendChild($objOption)
}

Add-EventHandler $w AvailableComputers onchange {
    param($object, $args)

    write-host "available computers list changed"
    $this | fl * | out-host
    $_ | gm | out-host
}

Add-EventHandler $w AvailableComputers onselect {
    write-host "available computers list changed"
    write-host ($_.GetType().FullName)
}
}

```

websearch.ps1

```

#
# Simple PowerShell search tool with an UI built in HTML
#
#
# Load the Web forms library
#
ipmo ./wpiawebforms

#
# HTML page to define the UI
#
$page = @"
<html>
<head><title>PowerShell Search Tool</title></head>
<body font="helvetica" bgcolor="lightblue">
  <center>
    <table style="width:100%;">
      <tr>
        <td>Path to search</td>
        <td><input type="text" id="path" size="60"/></td>
      </tr>
      <tr>
        <td>File Filter Pattern</td>
        <td><input type="text" id="filepattern" size="60"/></td>
      </tr>
      <tr>

```

```

        <td>Recursive search</td>
        <td><input id="recurse" type="checkbox" /></td>
    </tr>
    <tr>
        <td>Search Pattern</td>
        <td><input type="text" name="TextPattern" size="60" /></td>
    </tr>
    <tr>
        <td>Use regex</td>
        <td><input id="useregex" type="checkbox" /></td>
    </tr>
    <tr>
        <td>Only show first match</td>
        <td><input id="firstOnly" type="checkbox" /></td>
    </tr>
</table>
<button id="Search">Search</button>
<button id="Cancel">Cancel</button>
</center>
</body>
</html>
'@

#
# Invoke the form
#
Invoke-HtmlApplication -size 600,250 -title "PowerShell Search Tool" $page {

    # Set up element bindings to get values back from thr form
    Add-ElementPropertyBinding $webpage `
        FilePattern, Path, TextPattern, UseRegex, Recurse, FirstOnly

    # initialize the text fields
    $webpage.Path = $pwd
    $webpage.FilePattern = "*.ps1"
    $webpage.TextPattern = "function"

    # Set up the click handlers

    Add-EventHandler $webpage search onclick {

        function fixBool ($value) { "`$$($value -eq 'on')" }

        $cmd = "Get-ChildItem $($webpage.path) -Recurse: $(fixBool
($webpage.recurse)) -Filter '$($webpage.FilePattern)' |
        Select-String -SimpleMatch: $(fixBool (-not $webpage.useregex)) ``
        -Pattern '$($webpage.TextPattern)' ``
        -List: $(fixBool ($webpage.firstOnly))"
        "Command:`n$cmd" | out-host
        Invoke-Expression $cmd | out-host
        $form.close()
    }

    Add-EventHandler $webpage cancel onclick {
        $form.close()
    }
}

```

wpiaforms.psm1

```
#  
# WpiaWForms is a PowerShell Domain-Specific language for  
# building WinForm applications. It uses a declarative notation similar to  
# JavaFX from Sun. It is also similar in concept to XAML but uses language  
# notation instead of XML.#
```

```
# This is an example from Windows PowerShell in Action 2nd Edition  
#
```

```
Add-Type -AssemblyName System.Drawing  
Add-Type -AssemblyName System.Windows.Forms
```

```
function Point {new-object System.Drawing.Point $args}  
function Size {new-object System.Drawing.Size $args}
```

```
function Add-ControlBinding  
{  
    param (  
        [Parameter(mandatory=$true)]  
        [System.Windows.Forms.Control]  
        $control,  
        [Parameter(mandatory=$true)]  
        $property,  
        [Parameter(mandatory=$true)]  
        [System.Management.Automation.PSVariable]  
        $variable  
    )  
    $control.DataBindings.Add(  
        (new-object System.Windows.Forms.Binding $property, $variable,  
"value" ))  
    }  
}
```

```
function Form  
{  
    param (  
        $private:control,  
        [hashtable] $Properties = @{},  
        [switch] $Show,  
        [hashtable] $baseProperties = @{}  
    )  
  
    $private:events = @{}  
    $private:controls = $null  
    $private:items = $null  
  
    # if a baseProperties hashtable was provided, copy the keys  
    if ($baseProperties)  
    {  
        $baseProperties = @{} + $baseProperties  
        foreach ($private:prop in $properties.keys)  
        {
```

```

        $baseProperties[$prop] = $properties[$prop]
    }
}
else
{
    $baseProperties = $properties
}

foreach ($pn in "events", "controls", "items")
{
    if ($v = $baseProperties.$pn)
    {
        set-variable private:$pn $v
        $baseProperties.Remove($pn)
    }
}

$private:c = if ($baseProperties.Count) {
    new-object "Windows.Forms.$control" -Property $baseProperties
}
else
{
    new-object "Windows.Forms.$control"
}
if ($show)
{
    $c.SuspendLayout()
}

# execute the controls scriptblock if present and add the
# controls to the container...
if ($controls)
{
    [void] $c.controls.addrange(@(& $controls))
}

# execute the items scriptblock if present and add the
# items to the container...
if ($items)
{
    [void] $c.items.addrange(@(& $items))
    $c.SelectedIndex = 0
}

foreach ($private:en in $events.keys)
{
    $method = "add_$en"
    $c.$method.Invoke($events[$en])
}

# make sure the form is visible when run from a console
if ($control -eq "form") { $c.add_shown({ $this.Activate() }) }

if ($show)
{

```

```

        # Since we're going to show it immediately, place this form into the
        # Global MainForm so it can find itself.
        $c.ResumeLayout($false);
        $global:MainForm = $c
        [void] $c.ShowDialog()
    } else {
        $c
    }
}

function SetControlSize ($form, $stop) {
    $w = $f.Size.Width
    $h = $f.Size.Height
    $nw = $w-30
    $nh = $h-$stop.y-40
    $control.Size = size $nw $nh
}

function New-MenuStrip ($form, $menu)
{
    $ms = Form MenuStrip @{
        Location = point 0 0
        Name = "menuStrip"
        Size = Size 292 24
    }
    [void]$ms.Items.AddRange((&$menu))
    $form.Controls.Add($ms)
    $form.MainMenuStrip = $ms
}

function New-Menu($name, $menuitems)
{
    $menu = Form ToolStripMenuItem @{Text = $name}
    [void] $menu.DropDownItems.AddRange((&$menuitems))
    $menu
}

function New-MenuItem($name, $action)
{
    $item = Form ToolStripMenuItem @{Text = $name}
    [void] $item.Add_Click($action)
    $item
}

function Show-Form ($f) {
    [void] $f.ShowDialog()
}

```

WpiaWebForms.psm1

```

#
# WpiaWebForms is a PowerShell module that contains a number
# of utility functions useful from building PowerShell web forms.
# It requires the WpiaWinforms library
#

$mode = [System.Threading.Thread]::CurrentThread.ApartmentState
if ($mode -ne "STA")

```

```
{
    throw "This script can only be run when powershell is started with -sta"
}
```

```
Import-Module ./wpiaforms
```

```
function Get-Element
```

```
{
    <#
    .SYNOPSIS
    A utility to retrieve a control from an WebBrowser control id Id
    .DESCRIPTION
    This utility provides an easy way to retrieve a control
    from an HTML document control. Controls are retrieved based on the string
    in the Id property. For example, in the following
        <input type="text" id="path" size="60"/>
    the command to retrieve this would be
        Get-Element $document path
    .PARAMETER Document
    The document to search for the control
    .PARAMETER ControlId
    The control ID string to search for
    #>

    [cmdletbinding()]
    param(
        [parameter(mandatory=$true)]
            $Document,
        [parameter(mandatory=$true)]
            $ControlId
    )

    $Document.Document.GetElementById($ControlId)
}
```

```
function Add-EventHandler
```

```
{
    <#
    .SYNOPSIS
    A utility to add an event handler to a control a WebBrowser object
    .DESCRIPTION
    This utility provides an easy way to attach event handlers to
    controls in an HTML document. Controls are retrieved based on the string
    in the Id property. For example, in this HTML fragment
        <input type="button" id="Ok"/>
    the following command would bind the supplied scriptblock
    as the Click event handler for the control
        Add-EventHandler $webpage Ok OnClick { Write-Host "Hi!" }
    .PARAMETER Document
    The document to search for the control
    .PARAMETER ControlId
    The control ID string to search for
    .PARAMETER EventName
    The name of the event on the control to bind
    .PARAMETER Handler
    The scriptblock to use as the event handler
    #>
```

```

[cmdletbinding()]
param(
    [parameter(mandatory=$true)]
        $Document,
    [parameter(mandatory=$true)]
        $ControlId,
    [parameter(mandatory=$true)]
        $EventName,
    [parameter(mandatory=$true)]
        [scriptblock]
            $Handler
)

$control = $Document.Document.GetElementById($ControlId)
if (! $control)
{
    throw "control '$controlId' not found"
}
$control.AttachEventHandler($EventName, $handler)
}

#
# Retrieve the named element from the document object
#
function Get-ElementValue
{
    <#
    .SYNOPSIS
    A utility to retrieve the value from a control in a WebBrowser object
    .DESCRIPTION
    This utility provides an easy way to get the value from a control
    in a WebBrowser object. The control is selected based on the string
    in the Id property on the target control.
    .PARAMETER Document
    The document to search for the control
    .PARAMETER ControlId
    The control ID string to search for
    #>
    param(
        [parameter(mandatory=$true)]
            $Document,
        [parameter(mandatory=$true)]
            $ControlId
    )

    $element = $webpage.Document.GetElementById($ControlId)
    if (-not $element)
    {
        write-error "Error getting element '$elementname': no watching
element was found."
    }
    $element.DomElement.Value
}

function Set-ElementValue
{

```

```

<#
.SYNOPSIS
A utility to set the value for a control in a WebBrowser object
.DESCRPTION
This utility provides an easy way set the value of a control
in a WebBrowser object. The control is selected based on the string
in the Id property on the target control.
.PARAMETER Document
The document to search for the control
.PARAMETER ControlId
The control ID string to search for
.PARAMETER Value
The value to assign
#>
[CmdletBinding()]
param(
    [parameter(mandatory=$true)]
        $Document,
    [parameter(mandatory=$true)]
        $ControlId,
    [parameter(mandatory=$true)]
        $Value
)

$element = $Document.Document.GetElementById($ControlId)
if (-not $element)
{
    write-error "Error setting element '$elementname': no watching
element was found."
}
$element.DomElement.Value = $value
}

function Add-ElementPropertyBinding
{
    param(
        [parameter(mandatory=$true)]
            $WebPage,
        [parameter(mandatory=$true)]
            [string[]] $PropertyName
    )

    foreach ($name in $PropertyName)
    {
        Add-Member -InputObject $WebPage -MemberType ScriptProperty `
            -Name $name `
            { Get-ElementValue $WebPage $name }.GetNewClosure() `
            { Set-ElementValue $WebPage $name $args[0] }.GetNewClosure()
    }
}

function Invoke-HtmlApplication
{
    <#
    .SYNOPSIS
    Display an HTML document in a WinForms window using an
    initialization script to bind actions

```

```

    .DESCRIPTION
    This function creates a Windows Forms form containing an WebBrowser
control then
    sets the control's content to be the passed document
    .PARAMETER PageText
    The HTML text to display
    .PARAMETER InitializationScript
    A scriptblock that is executed after the WebBrowser control is
initialized.
    This scriptblock can be used to set control values or bind event
handlers.
    .PARAMETER Size
    The initial size to use for the page
    .PARAMETER Title
    The initial text to show in the forms title bar.
#>
[CmdletBinding()]
param (
    [Parameter(mandatory=$true)]
        $PageText,
    [Parameter(mandatory=$true)]
    [ScriptBlock]
        $InitializationScript,
    [Parameter(mandatory=$true)]
    [int[]]
        $Size = (800,600),
    $Title = "PowerShell Hypertext Application"
)
$WebPage = New-Object System.Windows.Forms.WebBrowser -Property @{
    dock = "fill"
    DocumentText = $pageText
}

$Form = New-Object System.Windows.Forms.Form -Property @{
    text = $title
    size = New-Object System.Drawing.Point $size
}
$form.Controls.Add($WebPage)

$WebPage.add_DocumentCompleted($InitializationScript.GetNewClosure())
$form.ShowDialog()
}

```