PowerTips MONTHLY

Part of the PowerShell.com reference library, brought to you by **Dr. Tobias Weltner**

Volume 8 January 2014

This Month's Topic:

Static .NET Methods

Dr. Tobias Weltner

Table of Contents

1. Static .NET Methods
2. Getting Documentation
3. Listing Static Members
4. Converting Types
5. Turning On Standby-Mode
6. Listing Available Culture IDs
7. Testing Numbers and Date
8. Get Localized Month Names
9. Listing Internet Explorer Cookies
10. Counting Special Characters
11. Getting Time Zones
12. Getting Character Ranges
13. Check for Wildcards
14. Get-Clipboard
15. Set-Clipboard
16. Playing Sound in PowerShell
17. Finding Days in Month
18. Finding Leap Years
19. Resolve Host Names
20. Renewing All DHCP Leases
21. Defining Return Values
22. Secret TimeSpan Shortcuts
23. Creating Relative Dates
24. Test Internet Connection
25. Finding Multiple RegEx Patterns
26. Translating Culture IDs to Country Names
27. Escape Regular Expressions
28. Extracting Icons
29. Getting SID of the Current User
30. Test Admin Privileges
31. Setting Mouse Position
32. Converting Date to WMI Date
33. Finding Built-In Cmdlets
34. Check for 64-Bit Environment
35. Converting Bitmaps to Icons
36. Showing MsgBox
37. ASCII Table
38. Check for Numeric Characters
39. Getting Significant Bytes
40. Permanent Changes to Environment Variables
41. Strongly-Typed Variables
42. Finding System Folders
43. Open File Exclusively
44. Removing File Extensions
45. Quickly Changing File Extensions
46. Identifying 64-Bit Environments

1. Static .NET Methods

Many .NET types contain useful static methods. Static methods can be invoked directly. To invoke a static method, you add two colons after the .NET type.

For example, the following line will generate a new GUID (unique identifier) that you can use to uniquely mark things:

```
PS> [System.Guid]::NewGuid().Guid
7cb953d9-c906-46e8-86e8-666a717e7bed
```

In this example, `[System.Guid]` is the .NET type you are using. .NET types are enclosed in brackets. NewGuid() is the method you are running. The method returns an object with a property called “Guid”:

```
PS> [System.Guid]::NewGuid()

Guid
----
cd87da91-9b4f-4c24-a18c-8acee41e3f3f
```

To get just the GUID number, the example accesses this property directly.

In this issue, we will present you a mixture of examples that all use static members. These examples cover only a fraction of useful .NET types but will give you a good understanding on how this technique works. Use it as a starting point to explore the other static members in the types.

2. Getting Documentation

.NET types that are part of the Microsoft .NET Framework come with excellent documentation. If you wanted to know more about the type System.GUID, simply navigate to your favorite Internet search engine, and enter the type name as search phrase.
When you add the additional search phrase "PowerShell", you will quickly find a vast number of articles on how others used the .NET type.

3. Listing Static Members

Once you know the name of a .NET type, you can use Get-Member to list all of its static properties and methods (members). This example will list all members found in the DateTime type:

```
PS> [DateTime] | Get-Member -Static


   TypeName: System.DateTime

Name            MemberType Definition
----            ---------- ----------
Compare         Method     static int Compare(datetime t1, datetime t2)
DaysInMonth     Method     static int DaysInMonth(int year, int month)
Equals          Method     static bool Equals(datetime t1, datetime t2), static bool
Equals(System.Object objA, System.Object objB)
FromBinary      Method     static datetime FromBinary(long dateData)
FromFileTime    Method     static datetime FromFileTime(long fileTime)
FromFileTimeUtc Method     static datetime FromFileTimeUtc(long fileTime)
FromOADate      Method     static datetime FromOADate(double d)
IsLeapYear      Method     static bool IsLeapYear(int year)
Parse           Method     static datetime Parse(string s), static datetime Parse(string s,
System.IFormatProvider provider), static datetime Parse...
ParseExact      Method     static datetime ParseExact(string s, string format, System.
IFormatProvider provider), static datetime ParseExact(string ...
ReferenceEquals Method     static bool ReferenceEquals(System.Object objA, System.Object objB)
SpecifyKind     Method     static datetime SpecifyKind(datetime value, System.DateTimeKind kind)
TryParse        Method     static bool TryParse(string s, [ref] datetime result), static bool
TryParse(string s, System.IFormatProvider provider, S...
TryParseExact   Method     static bool TryParseExact(string s, string format, System.
IFormatProvider provider, System.Globalization.DateTimeStyles ...
MaxValue        Property   static datetime MaxValue {get;}
MinValue        Property   static datetime MinValue {get;}
Now             Property   datetime Now {get;}
Today           Property   datetime Today {get;}
UtcNow          Property   datetime UtcNow {get;}
```

From the list, you can see that the type has a property called "UtcNow". So if you'd like to know the current UTC time, run this:

```
PS> [DateTime]::UtcNow

Monday, December 16, 2013 13:54:10
```

Likewise, you can use any of the methods (which always are followed by braces including the method arguments). For example, to convert ticks (the smallest time unit in Windows systems) to a real DateTime object, do this:

```
PS> [DateTime]::FromBinary(623577621536757899)

Saturday, January 15, 1977 08:35:53
```

4. Converting Types

You can easily convert types if you know the target type name. So if you wanted to convert a double number into an integer number, stripping the decimals and rounding the number, here is the solution:

```
PS> [Int]7.68
8
```

You can convert even more sophisticated things:

```
PS> [System.Version]'3.2.12.9'

Major  Minor  Build  Revision
-----  -----  -----  --------
3      2      12     9


PS> [System.Net.Mail.MailAddress]'Tobias Weltner<tobias.weltner@email.de>'

DisplayName           User                 Host          Address
-----------           ----                 ----          -------
Tobias Weltner        tobias.weltner       email.de      tobias.weltner@email.de


PS> [System.Net.Mail.MailAddress]'tobias.weltner@email.de'

DisplayName           User                 Host          Address
-----------           ----                 ----          -------
                      tobias.weltner       email.de      tobias.weltner@email.de


PS> ([System.Net.Mail.MailAddress]'tobias.weltner@email.de').Host
email.de
```

Author Bio

Tobias Weltner is a long-term Microsoft PowerShell MVP, located in Germany. Weltner offers entry-level and advanced PowerShell classes throughout Europe, targeting mid- to large-sized enterprises. He just organized the first German PowerShell Community conference which was a great success and will be repeated next year (more on www.pscommunity.de).. His latest 950-page "PowerShell 3.0 Workshop" was recently released by Microsoft Press.

To find out more about public and in-house training, get in touch with him at tobias.weltner@email.de.

5. Turning On Standby-Mode

To programmatically enter standby mode, you can use native .NET code like this:

```
function Invoke-Standby
{
  Add-Type -AssemblyName System.Windows.Forms
  [System.Windows.Forms.Application]::SetSuspendState(0,0,0) | Out-Null
}
```

6. Listing Available Culture IDs

Ever needed to get your hands on some specific culture IDs? Here is how you can create a list:

```
[System.Globalization.CultureInfo]::GetCultures('WindowsOnlyCultures')
```

Next, you can use a culture to output information formatted in that culture:

```
$c = [System.Globalization.CultureInfo]'ar-IQ'
[System.Threading.Thread]::CurrentThread.CurrentCulture = $c; Get-Date
```

You should note that the console character set is not able to display certain characters. You may want to run that command inside the PowerShell ISE or another Unicode-enabled environment.

7. Testing Numbers and Date

With a bit of creativity (and the help from the -as operator), you can create powerful test functions. These two functions test for valid numbers and valid DateTime information:

```
function Test-Numeric($Value) { ($Value -as [Int64]) -ne $null }
function Test-Date($Value) { ($Value -as [DateTime]) -ne $null }
```

If things get more complex, you can combine tests. This one tests for valid IP addresses:

```
function Test-IPAddress($Value) { ($Value -as [System.Net.IPAddress]) -ne $null -and ($Value -as
[Int]) -eq $null }
```

8. Get Localized Month Names

To get a list of day names, you could use this line:

```
[System.Enum]::GetNames([System.DayOfWeek])
```

However, this returns a culture-neutral list which is not returning the day in a localized (regional) form. To get the localized names, use this line instead:

```
0..6 | ForEach-Object { [Globalization.DatetimeFormatInfo]::CurrentInfo.DayNames[$_] }
```

To get month names, try this:

```
0..11 | ForEach-Object { [Globalization.DatetimeFormatInfo]::CurrentInfo.MonthNames[$_] }
```

9. Listing Internet Explorer Cookies

Have you ever wanted to find out who stored your information while you were surfing the Web? Check out your cookies folder:

```
Get-ChildItem ([Environment]::GetFolderPath("Cookies"))
Explorer ([Environment]::GetFolderPath("Cookies"))
```

And if you'd like, you could also select cookies based on content. The next line lists all cookies that contain the word "prefbin:"

```
Get-ChildItem ([Environment]::GetFolderPath("Cookies")) | Select-String prefbin -List
```

10. Counting Special Characters

Type conversion can help you count special characters in a text. For example, if you'd like to find out the number of tab characters in a text, you should do this:

```
@([Byte[]][char[]]$text -eq 9).Count
```

This line will convert the characters in $text into bytes and uses -eq to filter only the character code for tabs (which is the number 9). Next, @() ensures that the result is an array so that you can use Count to check the number of found characters.

11. Getting Time Zones

Here's a .NET call that returns all time zones:

```
[System.TimeZoneInfo]::GetSystemTimeZones
```

To find the time zone you are in, use this:

```
[System.TimeZoneInfo]::Local
([System.TimeZoneInfo]::Local).StandardName
```

Technical Editor Bio

Aleksandar Nikolic, Microsoft MVP for Windows PowerShell, a frequent speaker at the conferences (Microsoft Sinergija, PowerShell Deep Dive, NYC Techstravaganza, KulenDayz, PowerShell Summit) and the co-founder and editor of the PowerShell Magazine (http://powershellmagazine.com). He is also available for one-on-one online PowerShell trainings. You can find him on Twitter: https://twitter.com/alexandair

12. Getting Character Ranges

PowerShell's special ".." operator only supports numeric ranges:

```
1..10
```

You can use type conversion to get a range of letters:

```
[string][char[]](65..90)
[char[]](65..90) -join ','
```

13. Check for Wildcards

If you need to test for wildcards, you should try this .NET method:

```
$path = "c:\test\*"
[Management.Automation.WildcardPattern]::ContainsWildcardCharacters($path)
```

14. Get-Clipboard

If your PowerShell host uses the STA mode (which is default for PowerShell 3.0 and better), you can easily read clipboard content like this:

```
function Get-Clipboard {
  if ($Host.Runspace.ApartmentState -eq 'STA') {
        Add-Type -Assembly PresentationCore
        [Windows.Clipboard]::GetText()
    } else {
    Write-Warning ('Run {0} with the -STA parameter to use this function' -f $Host.Name)
  }
}
```

15. Set-Clipboard

If you are using Windows Vista or better, you can pipe text to clip.exe to copy it to your clipboard:

```
Dir $env:windir | clip
```

Here is yet another approach that you can use if your PowerShell host uses the STA mode:

```
function Set-Clipboard {
  param( $text )
  if ($Host.Runspace.ApartmentState -eq 'STA') {
        Add-Type -Assembly PresentationCore
        [Windows.Clipboard]::SetText($text)
    } else {
    Write-Warning ('Run {0} with the -STA parameter to use this function' -f $Host.Name)
  }
}

Set-Clipboard "Hello World"
```

16. Playing Sound in PowerShell

If you would like to catch a user's attention, you can make PowerShell beep like this:

```
[System.Console]::Beep()
[System.Console]::Beep(1000,300)
```

If you would like to catch a user's attention, you can make PowerShell beep like this:

```
[System.Media.SystemSounds]::Beep.Play()
[System.Media.SystemSounds]::Asterisk.Play()
[System.Media.SystemSounds]::Exclamation.Play()
[System.Media.SystemSounds]::Hand.Play()
```

17. Finding Days in Month

If you need to determine the days in a given month, you can use the static DaysInMonth() function provided by the DateTime type. As you can see, the days in a month are not always the same for every year:

```
[DateTime]::DaysInMonth(2009, 2)
[DateTime]::DaysInMonth(2008, 2)
```

18. Finding Leap Years

You will find that the DateTime type supports a number of static methods to check dates. For example, you can check whether a year is a leap year like this:

```
[DateTime]::IsLeapYear(1904)
```

19. Resolve Host Names

Have you ever needed to resolve a host name to find its IP address? Simply use a .NET method and wrap it as PowerShell function:

```
function Get-HostToIP($hostname) {
  $result = [System.Net.Dns]::GetHostByName($hostname)
  $result.AddressList | ForEach-Object {$_.IPAddressToString }
}
```

20. Renewing All DHCP Leases

Some WMI classes contain static methods. Static methods do not require an instance.

```
([WMIClass]"Win32_NetworkAdapterConfiguration").RenewDHCPLeaseAll().ReturnValue
```

21. Defining Return Values

Sorting or comparing IP addresses won't initially work because PowerShell uses alphanumeric comparison. However, you can compare or sort them correctly by casting IP addresses temporarily to the type System.Version:

```
$iplist = '1.10.10.1', '100.10.10.3', '2.10.10.230'
$iplist | Sort-Object
'*' * 40
$iplist | Sort-Object -Property { [System.Version]$_ }
```

22. Secret TimeSpan Shortcuts

Usually, to create time spans, you will use New-TimeSpan. However, you can also use a more developer-centric approach by converting a number to a TimeSpan type:

```
PS> [TimeSpan]100


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 0
Ticks             : 100
TotalDays         : 1,15740740740741E-10
TotalHours        : 2,77777777777778E-09
TotalMinutes      : 1,66666666666667E-07
TotalSeconds      : 1E-05
TotalMilliseconds : 0,01
```

This will get you a time span of 100 ticks, which is the smallest unit available to measure time. As PowerShell does support some hidden shortcuts for other units, this will create a time span of exactly five days:

```
PS> [TimeSpan]5d


Days              : 5
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 0
Ticks             : 4320000000000
TotalDays         : 5
TotalHours        : 120
TotalMinutes      : 7200
TotalSeconds      : 432000
TotalMilliseconds : 432000000
```

23. Creating Relative Dates

To create relative dates like “10 days ago,” there are two paths. Administrators often add or remove time spans created by New-TimeSpan:

```
(Get-Date) - (New-TimeSpan -Days 10)
```

Developers prefer object methods:

```
(Get-Date).AddDays(-10)
```

Both approaches will yield you the same result.

24. Test Internet Connection

Try this line if you would like to know whether a machine is currently connected to the Internet:

```
[Activator]::CreateInstance([Type]::GetTypeFromCLSID([Guid]'{DCB00C01-570F-4A9B-8D69-199FDBA5723B}')).IsConnectedToInternet
```

It returns $true when there is an Internet connection available.

25. Finding Multiple RegEx Patterns

PowerShell comes with great support for regular expressions but the -match operator can only find the first occurrence of a pattern. To find all occurrences, you can use the .NET RegEx type. Here is a sample:

$text = 'multiple emails like tobias.weltner@email.de and tobias@powershell.de in a string'

$emailpattern = '(?i)\b([A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4})\b'

```
$text = 'multiple emails like tobias.weltner@email.de and tobias@powershell.de in a string'
$emailpattern = '(?i)\b([A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4})\b'

([RegEx]$emailpattern).Matches($text) |
  ForEach-Object { $_.Groups[1].Value }
```

Note the statement "(?i)" in the regular expression pattern description. The RegEx object by default works case-sensitive. To ignore case, use this control statement.

26. Translating Culture IDs to Country Names

Have you ever wondered what a specific culture ID means? Here is a nifty function that will translate a culture ID to the full culture name:

```
[System.Globalization.CultureInfo]::GetCultureInfoByIetfLanguageTag('ar-IQ')
```

27. Escape Regular Expressions

PowerShell supports regular expressions in a lot of areas which is why the following code fails:

```
PS> 'c:\test\subfolder\file' -split '\'
parsing "\" - Illegal \ at end of pattern.
At line:1 char:1
+ 'c:\test\subfolder\file' -split '\'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], ArgumentException
    + FullyQualifiedErrorId : System.ArgumentException
```

Split expects a regular expression and fails when you use special characters like "\". To translate a plain text into an escaped regular expression text, try this:

```
[Regex]::Escape('\')
```

As you see, "\" is the RegEx escape character and therefore needs to be escaped:

```
'c:\test\subfolder\file' -split '\\'
('c:\test\subfolder\file' -split '\\')[-1]
```

28. Extracting Icons

To extract an icon from a file, use .NET Framework methods. Here is a sample that extracts all icons from all .exe files in your Windows folder (or one of its subfolders) and puts them into a separate folder:

```
Add-Type -AssemblyName System.Drawing

$folder = "$env:temp\icons"
mkdir $folder -ErrorAction 0 | Out-Null

Get-ChildItem $env:windir *.exe -ErrorAction 0 -Recurse |
ForEach-Object {
  $baseName = [System.IO.Path]::GetFileNameWithoutExtension($_.FullName)
  Write-Progress 'Extracting Icon' $baseName
  [System.Drawing.Icon]::ExtractAssociatedIcon($_.FullName).ToBitmap().Save("$folder\$BaseName.ico")
}

explorer.exe $folder
```

29. Getting SID of the Current User

To get the SID of the current user, try this:

```
PS> [Security.Principal.WindowsIdentity]::GetCurrent().User.Value

S-1-5-21-2649034417-1209187175-3910605729-1000
```

In fact, the method returns a wealth of additional information about your current access token:

```
PS> [Security.Principal.WindowsIdentity]::GetCurrent()



AuthenticationType : NTLM
ImpersonationLevel : None
IsAuthenticated    : True
IsGuest            : False
IsSystem           : False
IsAnonymous        : False
Name               : TobiasAir1\Tobias
Owner              : S-1-5-21-2649034417-1209187175-3910605729-1000
User               : S-1-5-21-2649034417-1209187175-3910605729-1000
Groups             : {S-1-5-21-2649034417-1209187175-3910605729-513, S-1-1-0, S-1-5-32-545,
                     S-1-5-4...}
Token              : 2276
UserClaims         : {http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name:
                     TobiasAir1\Tobias,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid:
                     S-1-5-21-2649034417-1209187175-3910605729-1000,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid:
                     S-1-5-21-2649034417-1209187175-3910605729-513,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid:
                     S-1-5-21-2649034417-1209187175-3910605729-513...}
DeviceClaims       : {}
Claims             : {http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name:
                     TobiasAir1\Tobias,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid:
                     S-1-5-21-2649034417-1209187175-3910605729-1000,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid:
```

```
                    S-1-5-21-2649034417-1209187175-3910605729-513,
                    http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid:
                    S-1-5-21-2649034417-1209187175-3910605729-513...}
Actor               :
BootstrapContext    :
Label               :
NameClaimType       : http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
RoleClaimType       : http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid
```

30. Test Admin Privileges

If you want to know whether you currently have Administrator privileges, use this function:

```
function Test-Admin {
  $identity = [Security.Principal.WindowsIdentity]::GetCurrent()
  $principal = New-Object Security.Principal.WindowsPrincipal $identity
  $principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)
}
```

It will return $true if you do have Administrator rights, else $false.

31. Setting Mouse Position

PowerShell can place the mouse cursor anywhere on your screen. Here's the code:

```
[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms') | Out-Null
[System.Windows.Forms.Cursor]::Position = New-Object System.Drawing.Point(500,100)
```

This would have placed the cursor at x=500 and y=100.

32. Converting Date to WMI Date

WMI (Windows Management Instrumentation, Get-WmiObject) uses a specific (DTMF) format for date and time. You can easily convert regular date and time information into this format:

```
$date = Get-Date
$wmidate = [System.Management.ManagementDateTimeConverter]::ToDmtfDateTime($date)
$wmidate
```

Likewise, you can convert WMI date and time information back to a regular date and time object. This code tells you when your system was rebooted the last time:

```
$os = Get-WmiObject -Class Win32_OperatingSystem
$os.LastBootUpTime
[System.Management.ManagementDateTimeConverter]::ToDateTime($os.LastBootupTime)
```

33. Finding Built-In Cmdlets

In times where cmdlets can originate from all kinds of modules, it sometimes becomes important to find out which cmdlets are truly built into PowerShell and which represent external dependencies.

One way of getting a list of built-in cmdlets is to temporarily open another runspace and enumerate its internal cmdlet list:

```
$ps = [PowerShell]::Create()
$ps.Runspace.RunspaceConfiguration.Cmdlets
$ps.Dispose()
```

34. Check for 64-Bit Environment

Here is a quick way of finding out your operating system architecture:

```
[System.Environment]::Is64BitOperatingSystem
[System.Environment]::Is64BitProcess
```

35. Converting Bitmaps to Icons

If you need a new icon and have no icon editor at hand, then you can take a bitmap file (create one with MS Paint if you must) and have PowerShell convert it into an icon file.

The function code also illustrates how you can open an Explorer window and select a file inside of it.

```
function ConvertTo-Icon {
  param(
    [Parameter(Mandatory=$true)]
    $bitmapPath,
    $iconPath = "$env:temp\newicon.ico"
  )

  Add-Type -AssemblyName System.Drawing

  if (Test-Path $bitmapPath) {
    $b = [System.Drawing.Bitmap]::FromFile($bitmapPath)
    $icon = [System.Drawing.Icon]::FromHandle($b.GetHicon())
    $file = New-Object System.IO.FileStream($iconPath, 'OpenOrCreate')
    $icon.Save($file)
    $file.Close()
    $icon.Dispose()

    explorer "/SELECT,$iconpath"
  } else { Write-Warning "$BitmapPath does not exist" }
}
```

36. Showing MsgBox

Ever wanted to display a dialog box from PowerShell rather than spitting out console text? Then try this function:

```
function Show-MsgBox {
  param
  (
    [Parameter(Mandatory=$true)]
    [String]$Text,

    [String]$Title = 'Message',

    [String]
    $Icon = 'YesNo,Information'
  )

  Add-Type -AssemblyName 'Microsoft.VisualBasic'

  [Microsoft.VisualBasic.Interaction]::MsgBox($text, $icon, $title)
}
```

It is really easy to use: simply tell it what to display:

```
Show-MsgBox -Text 'Reboot system?' -Title 'Warning' -Icon 'YesNoCancel,Question'
```

And this makes it a useful function:

```
$result = Show-MsgBox -Text 'Reboot system?' -Title 'Warning' -Icon 'YesNoCancel,Question'
if ($result -eq 'Yes') {
  Restart-Computer -Force
}
```

37. ASCII Table

Here's a simple way of creating an ASCII table through type casting:

```
32..255 | ForEach-Object {
  'Code {0} equals character {1}' -f $_, [Char]$_
}
```

Likewise, to get a list of letters, try this:

```
[Char[]](65..90)
-join [Char[]](65..90)
[Char[]](65..90) -join ','
```

38. Check for Numeric Characters

Try this if you need to check a single character and find out whether or not it is numeric:

```
[Char]::IsNumber('1')
[Char]::IsNumber('A')
```

The Char type has a bunch of other useful methods. Here is a list that you can try:

```
[Char] | Get-Member -Static
```

39. Getting Significant Bytes

If you need to split a decimal into bytes, you can use a function called ConvertTo-HighLow, which uses a clever combination of type casts to get you the high and low bytes:

```
function ConvertTo-HighLow {
  param(
    $number
  )
  $result = [System.Version][String]([System.Net.IPAddress]$number)
  $newobj = 1 | Select-Object Low, High, Low64, High64
  $newobj.Low = $result.Major
  $newobj.High = $result.Minor
  $newobj.Low64 = $result.Build
  $newobj.High64 = $result.Revision
  $newobj
}
```

40. Permanent Changes to Environment Variables

You can easily manage environment variables with the predefined env: drive. For example, to add a new environment variable, type this:

```
$env:launched = $true
```

However, these changes are visible only within the current PowerShell session because all changes are applied to the Process set of environment variables. As such, your changes would be visible to all other PowerShell scripts you launched from within the same PowerShell session. Changes are not permanent and will be discarded once you close PowerShell.

To make permanent changes, you will need to access the static .NET methods in the Environment class. This is how you read environment variables:

```
# environment variable for current user
[Environment]::GetEnvironmentVariable('Temp', 'User')

# environment variable for all users
[Environment]::GetEnvironmentVariable('Temp', 'Machine')
```

To change values, use SetEnvironmentVariable(). Note that you need full admin privileges to change machine-level environment variables because they affect all users.

```
[Environment]::SetEnvironmentVariable('Launched', $true, 'User')
[Environment]::SetEnvironmentVariable('Note', 'SomeContent', 'User')
```

Note: Changes you make to user-level and machine-level environment variables are visible in the env: drive only after you restart PowerShell. When PowerShell starts, it receives a copy of all environment variables, and this copy is not updated.

To delete an environment variable, set it back to an empty text:

```
[Environment]::SetEnvironmentVariable('Installed', '', 'User')
[Environment]::SetEnvironmentVariable('Note', '', 'User')
```

41. Strongly-Typed Variables

By assigning a type to a variable, the variable will use implicit conversion and always try and convert any new content to the type you assigned to the variable.

This adds an extra layer of security to your scripts because if you accidentally assign something to a variable that cannot be converted to the intended type, you will receive an error message.

```
[DateTime]$Date = Get-Date

$Date = '2014-01-12 19:22:10'
$Date

$Date = 'invalid data'
```

42. Finding System Folders

You may want to know where special folders such as MyPictures or Documents are located. The .NET class Environment provides a static method named GetFolderPath() which provides this information. To find the location of your desktop, for example, use this:

```
[Environment]::GetFolderPath('Desktop')
```

Simply specify an invalid keyword, and the exception will list all valid keywords:

```
PS> [Environment]::GetFolderPath('give me more!')
Cannot convert argument "folder", with value: "give me more!", for "GetFolderPath" to type
"System.Environment+SpecialFolder": "Cannot convert value
"give me more!" to type "System.Environment+SpecialFolder". Error: "Unable to match the identifier
name give me more! to a valid enumerator name.
Specify one of the following enumerator names and try again: Desktop, Programs, MyDocuments,
Personal, Favorites, Startup, Recent, SendTo, StartMenu,
MyMusic, MyVideos, DesktopDirectory, MyComputer, NetworkShortcuts, Fonts, Templates,
CommonStartMenu, CommonPrograms, CommonStartup,
CommonDesktopDirectory, ApplicationData, PrinterShortcuts, LocalApplicationData, InternetCache,
Cookies, History, CommonApplicationData, Windows,
System, ProgramFiles, MyPictures, UserProfile, SystemX86, ProgramFilesX86, CommonProgramFiles,
CommonProgramFilesX86, CommonTemplates,
CommonDocuments, CommonAdminTools, AdminTools, CommonMusic, CommonPictures, CommonVideos,
Resources, LocalizedResources, CommonOemLinks, CDBurning""
```

43. Open File Exclusively

To open a file in a locked state so no one else can open, access, read, or write the file, you can use the low-level .NET methods like this:

```
$path = "$env:temp\somefile.txt"    # MUST EXIST!

if ( (Test-Path $path) -eq $false)
{
  Set-Content -Value 'test' -Path $path
}

$file = [System.io.File]::Open($path, 'Open', 'Read', 'None')
$reader = New-Object System.IO.StreamReader($file)
$text = $reader.ReadToEnd()
$reader.Close()
Read-Host 'Press ENTER to release file!'
$file.Close()
```

This will lock a file and read its content. To illustrate the lock, the file will remain locked until you press ENTER.

44. Removing File Extensions

To remove file extensions, use .NET methods:

```
[System.IO.Path]::GetFileNameWithoutExtension('c:\test\report.txt')
```

45. Quickly Changing File Extensions

If you want to quickly exchange a file extension to create a "bak" backup version or generate a new file name for output, you should use the ChangeExtension() method:

```
$oldpath = 'c:\test\data.csv'
$newpath = [System.IO.Path]::ChangeExtension($oldpath, '.xls')
$oldpath
$newpath
```

46. Identifying 64-Bit Environments

You will find that one great advantage of 64-bit environments is the address width of 8 bytes instead of 4 bytes. You can use this to identify whether a script runs in a 32-bit or 64-bit environment. You should simply take a look at pointers and their address width:

```
if ([IntPtr]::Size -eq 8) { "Running in 64-bit subsystem" } else { "Running in 32-bit subsystem"
}
```