Excel macros with PowerShell

Alex Chaika Mon, May 1 2017 office, powershell 1

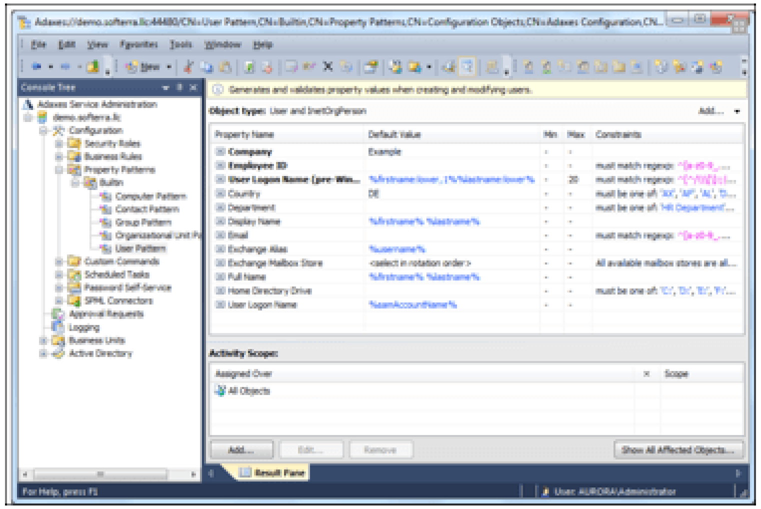
Almost everything you can do in the Microsoft Excel GUI can be done with PowerShell. Thus, you can use essentially use PowerShell to write Excel "macros."

Active Directory Management ADAXES

- Automated Provisioning
- Role-Based Delegation
- Active Directory Web UI
- Self-Password Reset
- Approval-Based Workflow
- Exchange & O365 Automation

FREE 30-day TRIAL

Advertisement



Generates and validates property values when creating and modifying users.

Object type: User and InetOrgPerson

Property Name	Default Value	Min	Max	Constraints
Company	Example	-	-	
Employee ID		-	-	must match regexp: ^[a-z0-9_...
User Logon Name (pre-Win...	%firstname:lower,1%%lastname:lower%	-	20	must match regexp: ^[^\/\[\]:;|...
Country	DE	-	-	must be one of: 'AX', 'AF', 'AL', 'D...
Department		-	-	must be one of: 'HR Department',...
Display Name	%firstname% %lastname%	-	-	
Email		-	-	must match regexp: ^[a-z0-9_...
Exchange Alias	%username%	-	-	
Exchange Mailbox Store	<select in rotation order>	-	-	All available mailbox stores are al...
Full Name	%firstname% %lastname%	-	-	
Home Directory Drive		-	-	must be one of: 'C:', 'D:', 'E:', 'F:'...
User Logon Name	%samAccountName%	-	-	

Activity Scope:

Assigned Over	Scope
All Objects	

About Latest Posts

Alex Chaika

Alex Chaika is a Microsoft Certified Solution Expert (MCSE) with more than 15 years of experience in IT systems engineering. He

Email *

Subscribe

Recently Active Members

Site Wide Activities [RSS]

currently focuses on PowerShell and VMware PowerCLI.

It is beyond the scope of this article to explain every available function because you could easily fill a book with this topic. Instead, I'll explain some basic concepts, which should be enough to create, modify, format, and save an Excel file using a PowerShell script.

Windows Server Performance Monitoring

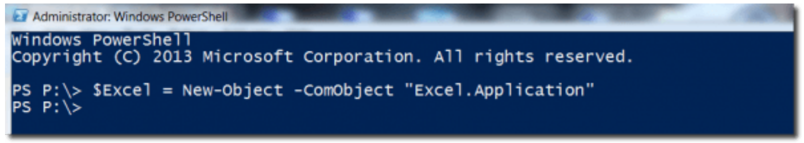
DATADOG

Create drag-and-drop dashboards to graph, analyze, correlate, and compare performance metrics and events in real-time.

Get started free!

Advertisement

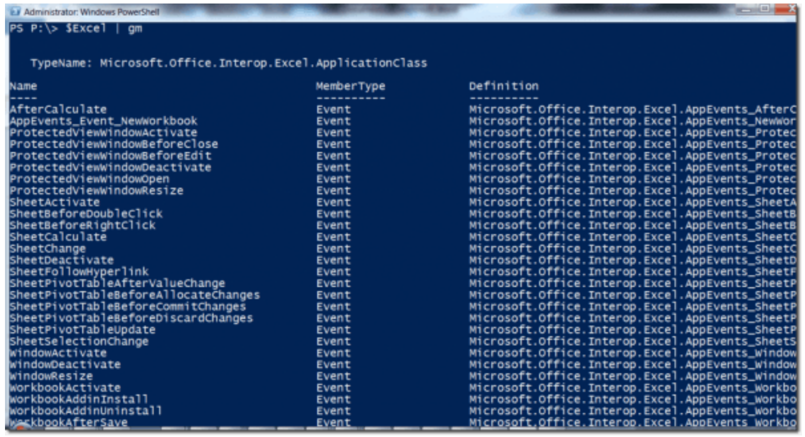
Notice that Excel has to be installed on the computer because you have to create an *Excel.Application* object in order to use its properties and methods:



```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS P:\> $Excel = New-Object -ComObject "Excel.Application"
PS P:\>
```

Creating an Excel COM application object

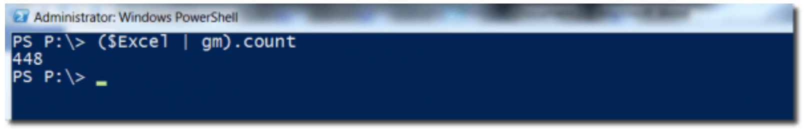
Now I'm going to explore what methods and properties the object provides:



```
PS P:\> $Excel | gm


   TypeName: Microsoft.Office.Interop.Excel.ApplicationClass

Name                                         MemberType            Definition
----                                         ----------            ----------
AfterCalculate                               Event                 Microsoft.Office.Interop.Excel.AppEvents_AfterC
AppEvents_Event_NewWorkbook                  Event                 Microsoft.Office.Interop.Excel.AppEvents_NewWor
ProtectedViewWindowActivate                  Event                 Microsoft.Office.Interop.Excel.AppEvents_Protec
ProtectedViewWindowBeforeClose               Event                 Microsoft.Office.Interop.Excel.AppEvents_Protec
ProtectedViewWindowBeforeEdit                Event                 Microsoft.Office.Interop.Excel.AppEvents_Protec
ProtectedViewWindowDeactivate                Event                 Microsoft.Office.Interop.Excel.AppEvents_Protec
ProtectedViewWindowOpen                      Event                 Microsoft.Office.Interop.Excel.AppEvents_Protec
ProtectedViewWindowResize                    Event                 Microsoft.Office.Interop.Excel.AppEvents_Protec
SheetActivate                                Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetA
SheetBeforeDoubleClick                       Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetB
SheetBeforeRightClick                        Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetB
SheetCalculate                               Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetC
SheetChange                                  Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetC
SheetDeactivate                              Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetD
SheetFollowHyperlink                         Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetF
SheetPivotTableAfterValueChange              Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetP
SheetPivotTableBeforeAllocateChanges         Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetP
SheetPivotTableBeforeCommitChanges           Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetP
SheetPivotTableBeforeDiscardChanges          Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetP
SheetPivotTableUpdate                        Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetP
SheetSelectionChange                         Event                 Microsoft.Office.Interop.Excel.AppEvents_SheetS
WindowActivate                               Event                 Microsoft.Office.Interop.Excel.AppEvents_Window
WindowDeactivate                             Event                 Microsoft.Office.Interop.Excel.AppEvents_Window
WindowResize                                 Event                 Microsoft.Office.Interop.Excel.AppEvents_Window
WorkbookActivate                             Event                 Microsoft.Office.Interop.Excel.AppEvents_Workbo
WorkbookAddinInstall                         Event                 Microsoft.Office.Interop.Excel.AppEvents_Workbo
WorkbookAddinUninstall                       Event                 Microsoft.Office.Interop.Excel.AppEvents_Workbo
WorkbookAfterSave                            Event                 Microsoft.Office.Interop.Excel.AppEvents_Workbo
```

Exploring the Excel COM application object resources

Just to give you an idea how powerful this object is, let's count the properties and methods:



```
PS P:\> ($Excel | gm).count
448
PS P:\> _
```

Number of properties and methods in the Excel application object

To create a new Excel workbook using PowerShell, we'll use the COM object we instantiated above. It is important to note that two methods exist here. Your PowerShell script can work in the background, so the process that changes the Excel sheet isn't visible. Alternatively, you can bring Excel to the screen to oversee all the changes your PowerShell script performs.

The Excel COM object's *Visible* property is responsible for this setting. If you set this variable to *$True*, all operations will be visible. If you set it to *$False*, all operations will occur in the background.

```
$Excel = New-Object -ComObject "Excel.Application"
$Excel.Visible = $true
$workbook = $Excel.Workbooks.Add()
```

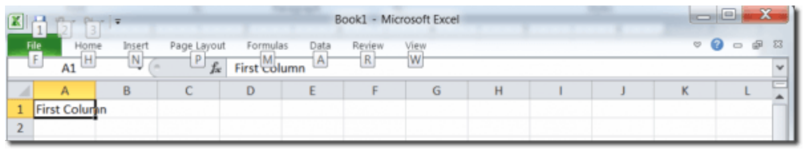
When you run this little script, an Excel window with a new workbook will pop up on the screen.

Since we've already created a workbook, let's add captions to the columns. To do so, we need to know the coordinates of the particular cell we are going to put the value into. To make it easier, let's add two variables to the script: one for the row ($Row) and one for the column ($Column).

This way, if we need to put a new value into the cell in the very first row and column, we just have to assign 1 to each of these variables. However, because an Excel workbook usually consists of several spreadsheets, we also need to tell our script which spreadsheet we want to change. You can do this by calling the spreadsheet either by name or by number. I'll use the name:

```
$Excel = New-Object -ComObject "Excel.Application"
$Excel.Visible = $true
$workbook = $Excel.Workbooks.Add()
$Sheet = $Workbook.Worksheets.Item("Sheet1")
$Row = 1
$Column = 1
$Sheet.Cells.Item($Row,$Column) = "First Column"
```

And here is my Excel spreadsheet with the column name created with PowerShell:



An Excel column name created with PowerShell

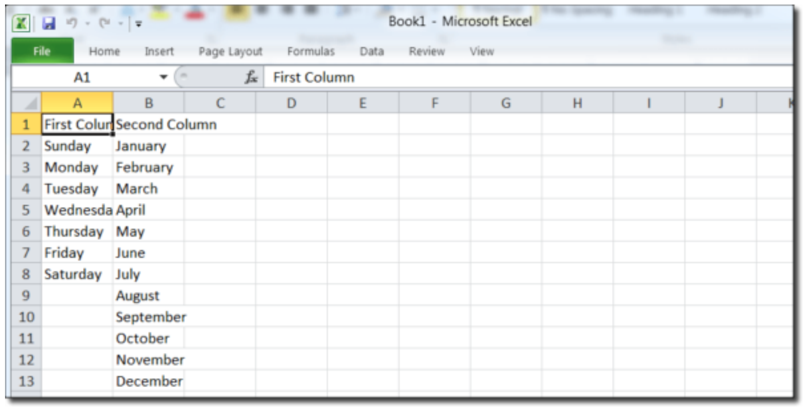
Now let's say I'd like to put the weekday names into the first column and the month names into the second one. Firstly, I need the caption for the second column. Then I'm getting the day and month names from the system.globalization.datetimeformatinfo object. I could create the arrays for the days and the months manually, but why not automate it with PowerShell?

I'm setting my $Column to 1 and the $Row to 2, so I can start putting the day names into the first column starting from the second row. Next, I'm using a *ForEach-Object* loop to iterate through the $DayNames array. To move down in the sheet, I'm incrementing the $Row value after each iteration. Then I do essentially the same operation with the month names:

```
$Column++
$Sheet.Cells.Item($Row,$Column) = "Second Column"
$SysDateObject = new-object system.globalization.date
$DayNames = $SysDateObject.Daynames
$MonthNames = $SysDateObject.MonthNames
$Column = 1
$Row = 2
$DayNames | %{
   $Sheet.Cells.Item($Row,$Column) = $_
   $Row++
  }
$Column = 2
$Row = 2
```

```
$MonthNames | %{
   $Sheet.Cells.Item($Row,$Column) = $_
   $Row++
  }
```

And here is what I get:



	A	B	C	D	E	F	G	H	I	J
1	First Colur	Second Column								
2	Sunday	January								
3	Monday	February								
4	Tuesday	March								
5	Wednesda	April								
6	Thursday	May								
7	Friday	June								
8	Saturday	July								
9		August								
10		September								
11		October								
12		November								
13		December								

A list of months and days in Excel created with PowerShell

Now I'd like to beef up my spreadsheet a little by adding column captions with a bold font and changing the background color of the cells. To do that, I'm defining the range of cells I'm going to change first, and then I just apply the font and color change to that range:

```
$Range = $Sheet.Range("A1:B1")
$Range.Interior.ColorIndex = 19
$Range.Font.ColorIndex = 11
```

```
$Range.Font.Bold = $True
```

As a final step, I'd like to format all my columns according to their width values. Because I'm going to apply this change to all cells in the current spreadsheet, I'll use the *UsedRange* method:

```
$Sheet.UsedRange.EntireColumn.AutoFit()
```

And finally I'm saving my workbook:

```
$Excel.ActiveWorkbook.SaveAs("C:\temp\myworkbook.xlsx'
```

Below is the complete script we've just written and a screenshot of the Excel spreadsheet we've created and formatted with PowerShell.

```
$Excel = New-Object -ComObject "Excel.Application"
$Excel.Visible = $true
$workbook = $Excel.Workbooks.Add()
$Sheet = $Workbook.Worksheets.Item("Sheet1")
$Row = 1
$Column = 1
$Sheet.Cells.Item($Row,$Column) = "First Column"
$Column++
$Sheet.Cells.Item($Row,$Column) = "Second Column"

$SysDateObject = new-object system.globalization.date
```

```
$DayNames = $SysDateObject.Daynames
$MonthNames = $SysDateObject.MonthNames
$Column = 1
$Row = 2
$DayNames | %{
   $Sheet.Cells.Item($Row,$Column) = $_
   $Row++
  }
$Column = 2
$Row = 2
$MonthNames | %{
   $Sheet.Cells.Item($Row,$Column) = $_
   $Row++
  }

$Range = $Sheet.Range("A1:B1")
$Range.Interior.ColorIndex = 19
$Range.Font.ColorIndex = 11
$Range.Font.Bold = $True
$Sheet.UsedRange.EntireColumn.AutoFit()
$Excel.ActiveWorkbook.SaveAs("C:\temp\myworkbook.xls
```

Excel spreadsheet created by PowerShell

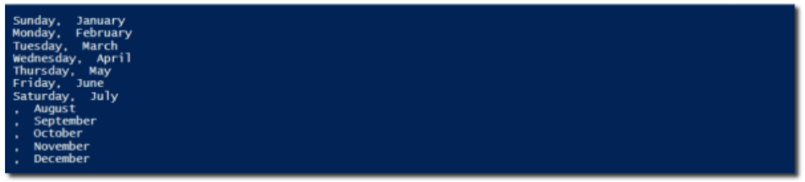
Let's try to access and use the data from the spreadsheet. As in my previous example, we are going to create an Excel COM object and then use its methods and properties. The only difference is that instead of using the *Add* method, we have to use the *Open* method to open the workbook we created before. Now I am ready to access and manipulate the date in the spreadsheet.

```powershell
$Excel = New-Object -ComObject "Excel.Application"
$Excel.Visible = $true
$workbook = $Excel.Workbooks.Open("C:\temp\myworkbook
$Sheet = $Workbook.Worksheets.Item("Sheet1")
```

Next, I'm establishing the variables for the rows and the columns, and then I use those variables for loading the data from the spreadsheet. To make sure that I get all the data from the table, I'm using a *do-while* loop to test the value in the second column. The loop stops when this value become $NULL.

```powershell
$Excel = New-Object -ComObject "Excel.Application"
$Excel.Visible = $true
$workbook = $Excel.Workbooks.Open("C:\temp\myworkbook
$Sheet = $Workbook.Worksheets.Item("Sheet1")
$Row = 2
$1stColumn = 1
$2ndColumn = 2
Do {
$Day = $Sheet.Cells.Item($Row, $1stColumn).Value()
$Month = $Sheet.Cells.Item($Row, $2ndColumn).Value()
Write-Host "$Day,  $Month"
$Row++
} While ($Sheet.Cells.Item($Row,$2ndColumn).Value() -
```

And here is the output of the script:



```
Sunday,  January
Monday,  February
Tuesday,  March
Wednesday,  April
Thursday,  May
Friday,  June
Saturday,  July
,  August
,  September
,  October
,  November
,  December
```

Reading data from an Excel Spreadsheet with PowerShell

Note that you can also address a particular cell by its coordinates. For instance, if you have to read the contents of the cell in the middle of the spreadsheet, you don't have to go all the way down. Instead, you can just directly access the cell using the row and column numbers.

For example, to read the value in row 3, column 2, the command would look like this:

```
$Value = $Sheet.Cells.Item('3', '2').Value()
```

Sometimes this ability can save you many iterations, as when working with CSV files from PowerShell. The advantage of working with CSV files instead of Excel sheets is that Excel doesn't have to be installed on the computer. However, working with the Excel COM object and PowerShell certainly give you more options, and the most notable one is the ability to target cells directly.

On the other hand, PowerShell is certainly a much more powerful language than Visual Basic for Applications (VBA). If you are familiar with PowerShell you might be faster to automate a task with PowerShell than with a native Excel macro.

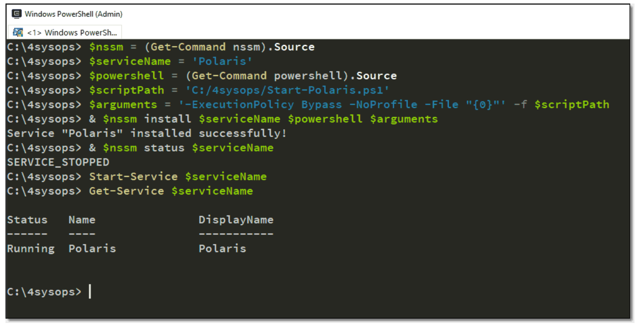
In my next post, I will show you to edit Word documents with PowerShell.

Join the 4sysops PowerShell group!

Share 15 1+

Users who have LIKED this post:

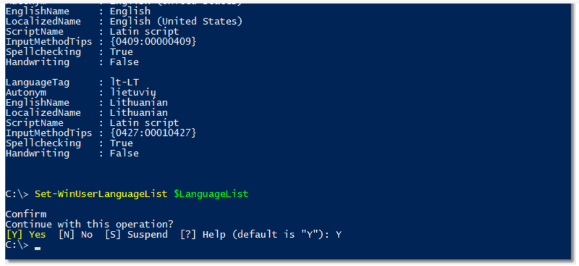
Related Posts



```
C:\4sysops> $nssm = (Get-Command nssm).Source
C:\4sysops> $serviceName = 'Polaris'
C:\4sysops> $powershell = (Get-Command powershell).Source
C:\4sysops> $scriptPath = 'C:/4sysops/Start-Polaris.ps1'
C:\4sysops> $arguments = '-ExecutionPolicy Bypass -NoProfile -File "{0}"' -f $scriptPath
C:\4sysops> & $nssm install $serviceName $powershell $arguments
Service "Polaris" installed successfully!
C:\4sysops> & $nssm status $serviceName
SERVICE_STOPPED
C:\4sysops> Start-Service $serviceName
C:\4sysops> Get-Service $serviceName

Status   Name               DisplayName
------   ----               -----------
Running  Polaris            Polaris


C:\4sysops> |
```

How to run a PowerShell script as a Windows servic...

```
C:\> $LanguageList.Add("lt-LT")
C:\> $LanguageList


LanguageTag     : en-US
Autonym         : English (United States)
```



```
EnglishName      : English
LocalizedName    : English (United States)
ScriptName       : Latin script
InputMethodTips : {0409:00000409}
Spellchecking    : True
Handwriting      : False

LanguageTag      : lt-LT
Autonym          : lietuvių
EnglishName      : Lithuanian
LocalizedName    : Lithuanian
ScriptName       : Latin script
InputMethodTips : {0427:00010427}
Spellchecking    : True
Handwriting      : False



C:\> Set-WinUserLanguageList $LanguageList

Confirm
Continue with this operation?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): Y
C:\> _
```

Adding and removing keyboard languages with PowerS...

1 COMMENT

David

9 months ago

Alex:

I have a quick question. Hopefully, you can assist me or at least point me to the right solution:

I've an urgent project to complete , where I have to look/search for specific value/number (=>100) in column "P" in massive multiple excel files that I can point to (data sets). After that, I need to print out/save a report with folder name where that particular value was found. I was wondering if Powergrep can do this type of search. I'm very short on time.I've tried to play with Powergrep around, but couldn't find an option where I can look at specific column in Excel files.Here is an example of folder structure and what I need to look for and what is my output should be:

Looking at column "P" in excel (column called "Temperature") for value that is =>100

C:\Folder\ Folder1 Folder2

Folder1 Excel1 Excel2 Excel3

Folder2 Excel10 Excel200

Output: The value was found @: Folder2

Any help will be much appreciated!

Thanks! David

1+

REPLY

Leave a reply

Your email address will not be published. Required fields are marked *

Visual Text

Notify me of followup comments via e-mail

Name *	Email *	Website

POST COMMENT

Blog Authors (Recently active)

Brandon Olin

Mohamed A. Waly

Bryce McDonald

Dan Franciscus

Ruben Zimmermann

Reviews

Install internalized Chocolatey packages from your offline repository
Tue, May 29 2018

Condusiv V-locity v7: I/O reduction and optimization for your virtual machines
Tue, May 8 2018

New in Altaro VM Backup v7.6
Wed, May 2 2018

NTFS Permissions

Popular Posts

Office 2016 installation
Tue, Nov 10 2015

Disable Windows 10 Update in the Registry and with PowerShell
Mon, Aug 17 2015

Automatically log off idle users in Windows
Thu, Mar 26 2015

The PowerShell function - Parameters, data types, return values
Thu, Feb 12 2015

Forum Topics

Powershell script not running when attached to an event and set on schedule
Tue, May 29 2018

Free Internet-Proxy and Web-Filtering service
Fri, May 25 2018

Cross-forest CA - clients not able to see templates
Thu, May 10 2018

Outlook 2016 Mac search missing emails and tasks -

Baki Onur Okutucu

Michael Pietroforte

Karim Buzdar

Timothy Warner

Auditor: No-Nonsense file system security auditing and reporting
Wed, Apr 25 2018

Create local and remote NTFS permissions reports with G-TAC FolderSecurityViewer
Wed, Apr 4 2018

Software deployment with PowerShell and Chocolatey
Tue, Mar 13 2018

3CX: Enterprise-class unified communications designed for ease of use
Thu, Mar 8 2018

Reset a Windows 10 password
Thu, Oct 30 2014

Disable Windows Update in Windows 10
Mon, Oct 13 2014

Use DISM to slipstream updates
Fri, Mar 21 2014

Managing inactive clients in SCCM 2012
Wed, Oct 16 2013

OutlookSearchRepair tool
Thu, May 10 2018

Defending against PowerShell Attacks - Security Talk by Lee Holmes (Microsoft)
Fri, May 4 2018

Issues when using Domain credentials for Wifi Access.
Fri, May 4 2018

NTFS versus ReFS for a File Server?
Wed, Apr 25 2018

Windows Server 2016 cluster with multiple SANs?
Wed, Apr 25 2018

What do you like to read about here?

Mon, Apr 16 2018