Windows **PowerShell**

www.scriptrunner.com

STRINGS AND EXPRESSIONS

Embedding of a variable in a string
```
"The command is $Command!"
```

{} must be used here to delimit it from the colon
```
"${Command}: executed successfully"
```

The subexpression must be parenthesized in $( )
```
"$($Result.Count) objects in result set"
```

Use of the format operator
```
Get-Process | % { "{0,-40} uses {1:0,000.00}MB" -f $_.Name, ($_.ws/1MB)
}
```

Execute a string as a command
```
$Command = "Get-Service a*"
$Command += "| where status -eq ‚Running'"
$Result = Invoke-Expression $Command
$Command | Format-List
$Result | Format-List
```

POWERSHELL DATA TYPES

Numeric types
```
[byte]
[int]
[long]
[single]
[double]
```

Generate random number between 1 and 49 and store in variable $x
```
[byte] $x = Get-Random -Minimum 1 -Maximum 49
```

Character types
```
[char]
[string]
```

Boolean and date types
```
[bool]
[DateTime]
```

Store current date in variable $d
```
[Datetime] $d = Get-Date
```

Object sets
```
[Array]
[Hashtable]
```

Store list of services starting with "a" in variable $services
```
[Array] $services = Get-Service a*
```

More complex data structures
```
[XML]
[WMI]
[ADSI]
```

A complete list of TypeAccelerators is accessible with:
```
[psobject].Assembly.GetType('System.Management.Automation.
TypeAccelerators')::Get
```

COMPARISON OPERATORS

Compare case in-sensitive	Compare case sensitive	Meaning
-lt -ilt	-clt	Less than
-le -ile	-cle	Less or equal
-gt -igt	-cgt	Greater than
-ge -ige	-cge	Greater or equal
-eq -ieq	-ceq	Equal
-ne -ine	-cne	Not equal
-like -ilike	-clike	Similarity between strings, use of wildcards (* and ?) possible
-notlike -inotlike	-cnotlike	No similarity between strings, use of wildcards (* and ?) possible
-match -imatch	-cmatch	Compare with regular expression
-notmatch -inotmatch	-cnotmatch	Does not match regular expression
-is	–	Type comparison, e.g. (Get-Date) -is [DateTime]
-in -contains	–	Is included in set
-notin -notcontains	–	Is not included in set

For logical conjunction, -and, -or as well as -not (alias !) are used
Example: ((1MB + 250 + $ a) -gt 2000KB) -and! ($ A -le 2KB)
KB, MB, GB, TB, and PB are valid units for memory sizes.

USING .NET FRAMEWORK CLASSES

Access to static members
```
[System.Environment]::MachineName
[System.Console]::Beep(800, 500)
```

Instantiation and access to instance members
```
$b = New-Object System.Directoryservices.DirectoryEntry
("WinNT://MyServer/ScriptRunner")
$b.FullName
$b.Description = "PowerShell Automation"
$b.SetInfo()
```

Load and use additional assembly
```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.VisualBa-
sic")
$input = [Microsoft.VisualBasic.Interaction]::InputBox("Please enter your
name!","Title")
```

CONFIGURING POWERSHELL

Set-ExecutionPolicy Unrestricted	Allow all PowerShell scripts
Set-ExecutionPolicy RemoteSigned / AllSigned	Only allow signed PowerShell scripts
Enable-PSRemoting -SkipNetworkProfileCheck	Enable PowerShell remote access for this machine - even if there are public networks
(Get-Host).PrivateData.ErrorBackgroundcolor ="White"	Change background colour for error messages (increases contrast of red characters)

USING MODULES

Get-Module	List activated modules
Get-Module -ListAvailable	List all installed modules
Import-Module	Enable local module for current session
Find-Module	Search modules in PowerShell Gallery
Install-Module	Download and install modules from PowerShell Gallery
Update-Module	Update module

with kind permission of Dr. Holger Schwichtenberg: www.dotnet-doktor.de

INPUT AND OUTPUT COMMANDLETS

Format-Table (ft)	Table output
Format-List (fl)	Detailed list
Format-Wide (fw)	Multi-column list
Out-Host (oh)	Output to consoles with colour options and paging option
Out-GridView (ogv)	Table with filtering and sorting options
Out-File	Save to file
Out-Printer (lp)	Send to printer
Out-Clipboard	Send to clipboard
Out-Speech	Speech output (requires module "PSCX")
Out-Null	Objects in pipeline are not passed on
Read-Host	Read from console
Import-CSV Export-CSV	Import / export CSV file
Import-CLIXML Export-CLIXML	Import / export XML file

User defined table output
```
Get-Process | ft @{Label="Nr"; Expression={$_.ID}; Width=5},
@{Label="Name"; Expression={$_.Processname}; Width=30},
@{Label="Memory MB"; Expression={$_.WorkingSet64 / 1MB};
Width=7; Format="{0:00000.0}"}
```

CONFIGURING AND USING NETWORKS

Get-NetAdapter	List network cards (also virtual ones)
Get-NetAdapterBinding	Properties of a network connection
Set-NetIPInterface	Enable or disable DHCP
New-NetIPAddress Remove-NetIPAddress	Set or remove static IP address
Set-DnsClientServerAd-dress	Set or remove DNS server
Remove-NetRoute	Remove gateway from network connection
Resolve-DnsName	Resolve DNS name
Enable-NetFirewallRule Disable-NetFirewallRule	Enable or disable a Windows Firewall rule
Test-Connection	Perform a ping
Send-MailMessage	Send email
Invoke-WebRequest	HTTP request
New-WebServiceProxy	Create a proxy for SOAP-based service
Export-ODataEndpoint-Proxy	Create proxy for OData-based service

ACCESS TO WMI

List of all WMI classes from a namespace of a computer
```
Get-CimClass -Namespace root/cimv2 -Computer MyServer
```

List all instances of a WMI class on a computer
```
Get-CimInstance Win32_LogicalDisk -Namespace root/cimv2 -Computer
MyServer
```

WQL query on a computer
```
Get-CimInstance -Query "Select * from Win32_Networkadapter where
adaptertype like '%802%'" -Computer MyServer
```

Access to an instance and change to the instance
```
$c = Get-CimInstance Win32_LogicalDisk -Namespace root/cimv2 -Filter
"DeviceID='C:'" -Computer MyServer
$c.VolumeName = "System"
Set-CimInstance $c
```

Alternatively with old WMI commandlets
```
$c = [WMI] "\\MyServer\root\cimv2:Win32_LogicalDisk.DeviceID='C:'"
$c.VolumeName = "System"
$c.Put()
```

Calling a WMI method
```
Invoke-CimMethod -Path "\\MyServer\root\cimv2:Win32_Computersys-
tem.
Name=MyServer" -Name "Rename" -ArgumentList "MyNewServer"
```

IMPORTANT NAVIGATION COMMANDLETS

Get-PSDrive	List of drives
Get-Location (pwd)	Retrieve current location
Set-Location (cd)	Set current location
Get-Item (gi)	Get an element
Get-ChildItem (dir, ls, gci)	List all subelements
Get-Content (type, cat, gc)	Retrieve content of an element (e.g. file content)
Set-Content (sc)	Set element content
Add-Content (ac)	Add element content
New-Item (ni, mkdir)	Create an element (branch or leaf)
Get-ItemProperty (gp)	Retrieve attribute
Set-ItemProperty (sp)	Define an attribute of an element, create one if necessary
Remove-Item (del, ri, rmdir, rm, erase)	Delete element
Move-Item (move, mv)	Move element
Copy-Item (copy, cp, cpi)	Copy element
Rename-Item (rni, ren)	Rename element

With Navigation Commandlets it's not only possible to operate on the file system
(drives A:, B:, C: etc.), but also on other flat and
hierarchical sets like Registry (HKLM:, HKCU:),
Environment (ENV:), Certificate (CERT:), Active Directory (AD:),
IIS-Webserver (IIS:) etc.:

```
Dir HKLM://software/
New-Item HKLM://software/ScriptRunner
RD HKLM://software/ScriptRunner
```

POWERSHELL CLASSES

Implementation of the PowerShell class
```
class User
{
# Properties
[int] $ID
[string] $Name
hidden [Datetime] $CreatedOn
# Static Property
static [Int64] $Count

# Constructor
User([int] $newid, [string] $name)
{
$this.ID = $newid
$this.Name = $name
$this.CreatedOn = Get-Date
[User]::Count = [User]::Count + 1
}
# Method
[string] GetInfo([bool] $verbose, [string] $separator)
{
[string] $a = "$($this.ID)$separator$($this.Name)";
if ($verbose) { $a += "$separator$($this.CreatedOn)" }
return $a
}
# Static Method
static [string] GetCount()
{
return "In total $([User]::Count) users!"
}}
```

Use of the PowerShell class
```
$b = [User]::new(123,"Matt Scripter")
$b.ID = 1
$b.Name = "Matt"
$b.GetInfo($true,";")
[User]::GetCount()
```

ONLINE RESOURCES

technet.microsoft.com/scriptcenter

blogs.msdn.com/PowerShell

github.com/ScriptRunner

scriptrunner.com/blog

scriptrunner.com

GET HELP

Get-Command Get- *	All commands with „Get-"
Get-Command | where module -like "ActiveDirectory* | ft Name, Module	All commands of a module
Get-Alias	Show all aliases
Get-Help Stop-Process -full	Full help content for a command
Get-Help about	List all „About" documents
Get-Help about_WMI	Show help for WMI
Get-Service | Get-Member	Show all properties and methods of the result objects

POWERSHELL SCRIPTING LANGUAGE

Condition
```
if ((Get-Date).Year -le 2014) { "Old" } else { "New" }
```

Loops
```
for($i = 1; $i -le 10; $i++) { $i }
while($i -le 10) { $i; $i++ }
do { $i; $i++ } while ($i -le 10)
foreach ($p in (Get-Process iexplore)) { $p.Kill() }
```

Subroutines with mandatory parameters and optional parameters
```
function Get-DLL([Parameter(Mandatory=$true)][string]$root,
[string]$filter = "*")
{
return Get-ChildItem $root -Filter "$filter.dll"
}
Get-DLL c:\Windows\System32
```

Comment
```
# This is a comment
```

ACTIVE DIRECTORY (AD) *requires Module Active Directory

Get-ADObject	Retrieve arbitrary objects from AD
Get-ADUser Get-ADGroup Get-ADOrganizationalUnit Get-ADDomain Get-ADComputer	Retrieve particular AD elements
Set-ADObject Set-ADUser Set-ADGroup Set-ADComputer	Set properties for an object
New-ADUser New-ADGroup New-ADOrganizationa-lUnit	Create new AD object
Remove-ADObject	Delete AD object
Rename-ADObject	Rename AD object
Move-ADObject	Move AD object
Set-ADAccountPassword	Set password
Get-ADGroupMember	List group members of an AD group
Add-ADGroupMember	Add member to an AD group
Remove-ADGroupMember	Remove member from an AD group

PROCESSES, SERVICES, EVENTS, PERFORMANCE INDICATORS

Get-Process	Running processes
Start-Process Stop-Process	Start/terminate process
Wait-Process	Wait for process to terminate
Get-Service	Windows system services
Start-Service Stop-Service Suspend-Service Resume-Service	Change service state
Get-Win Event	Event log entries
New-WinEvent	Create entry in event log
Limit-EventLog	Set size for event log
Get-Counter	Retrieve important performance indicators
Get-Counter -ListSet *	List all performance indicators
Get-Counter -Counter "\Processor(_Total)\% Processor Time"	Retrieve particular performance indi-cator

ScriptRunner®

The #1 for PowerShell Management

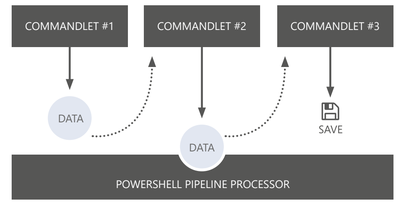
TURN PAGE

PIPELINING

Any number of commandlets can be joined using the pipe symbol |.
```
Get-Service a* | Where-Object {$_.status -eq "running"} |
Out-File c:\temp\runningservices.txt
```

Alternatively, you can store intermediate results in variables starting with $.
```
$services = Get-Service a* | Where-Object {$_.status -eq "running"}
$services | Out-File c:\temp\runningservices.txt
```

The pipeline forwards NET objects. Forwarding is asynchronous
(except from some "blocking" commandlets like the sort object)



EXAMPLE:
```
Get-Service a* | Where-Object { $_ .status-eq "running" } | Out-File c:\file-
name.txt
```

Commandlet #1:
```
Get-Service a*
```
Object of type: System. ServiceProcess. ServiceController

Commandlet #2 - Selection:
```
Where-Object { $_ .status-eq "running" }
```

Commandlet #3 - Storage in file system
```
Out-File c:\filename.txt
```

IMPORTANT PIPELINING COMMANDLETS

Where-Object (where, ?)	Filter using conditions
Select-Object (select)	Truncate result set from its start/end reduction of object attributes, respectively
Sort-Object (sort)	Sort objects
Group-Object (group)	Group objects
Foreach-Object { $_... } (%)	Loop over all objects
Get-Member (gm)	Print metadata (reflection)
Measure-Object (measure)	Calculation: -min -max -sum -average
Compare-Object (compare, diff)	Compare two sets of objects

OBJECT-ORIENTED ACCESS TO PIPELINE OBJECTS

Number of objects in pipeline
```
(Get-Service | where { $_.status -eq "Running" }).Count
```

Print particular properties of pipeline objects
```
(Get-Date).DayOfWeek
(Get-Process).Name
(Get-Process | sort ws -desc)[0].Name
```

Method call in all pipeline objects
```
(Get-Process iexplore | sort ws -desc).Kill()
```

SOFTWARE INSTALLATION

Get-PackageProvider	List all types of packages that PowerShell is able to download and install (e.g. Nuget, Chocolatey, MSI)
Install-PackageProvider choco-latey	Install new package provider (of a new package type)
Register-Packagesource -Name chocolatey -Provider chocolatey -Trusted -Location http://chocolatey.org/api/v2/ -Verbose	Register new package source for Chocolatey.org
Get-PackageSource	List all registered package sources
Find-Package -Name chrome -Source chocolatey	Search package source for software by specific word in title
Install-Package googlechrome -Source chocolatey	Install software package "GoogleChrome" from a particular package source
Get-Package	List all installed packages
Uninstall-Package googlechrome	Uninstall a software package

Dashboard

Actions

Queries

Targets

Credentials

Scripts | Cmdlets

Delegation

Automation

Settings

Actions

Global Filters: AzureAD× Tags | Azure Team

Filter: 67 -> 3

Search...

☑1	!	Sched.	Name	Targets	Favs	Last run	Recently	Status	Tags
☑	↱		Enable/Disable User	AD Target on SR Host	★★★★☆	Jul 6, 2019			ActiveDirectory

ACTIONS

- An **ACTION** is a policy set for the controlled execution of a PowerShell script on a target system. An action can start interactively, scheduled or externally.
- The **ACTION** determines who is allowed to start which script, in which context it is to be executed, and which inputs are necessary.

QUERIES

- A **QUERY** is a dynamic element which allows interactive search in the Active Directory, via script as well as from lists and files.
- **QUERIES** can be cascaded and start automatically, can run in real-time, be scheduled, or run from cache.

TARGETS

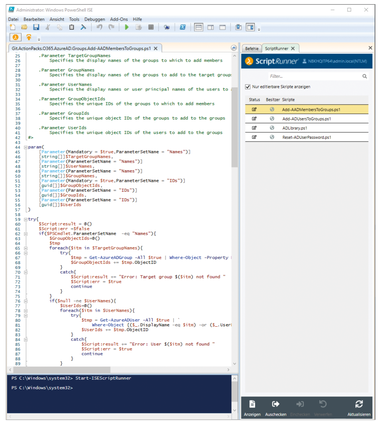
- A **TARGET** system is the location where the scripts are executed in run-time with their policy settings. A collection of targets allows parallel script execution.
- PowerShell scripts can be executed locally on ScriptRunner and with remoting in on-prem, hybrid- or cloud infrastructures.

CREDENTIALS

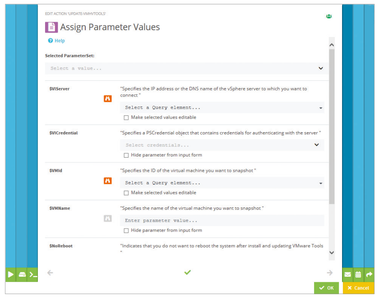
- A **CREDENTIAL** is an administrative, privileged account. Only these accounts are authorized to execute scripts on the target systems.
- **CREDENTIALS** can be inserted in scripts with placeholders of type [PSCredential] at runtime.

SCRIPTS

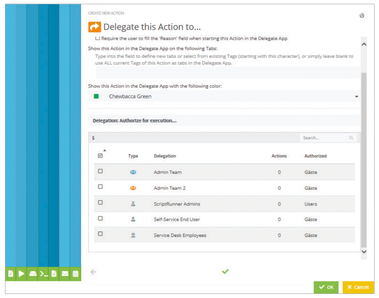
- PowerShell **SCRIPTS** are centrally managed and used in actions and are also used to automatically generate Web Forms in the browser apps.
- **SCRIPTS** can be run as main script, query script or as a reusable function library. An own hashtable is available to use in scripts.



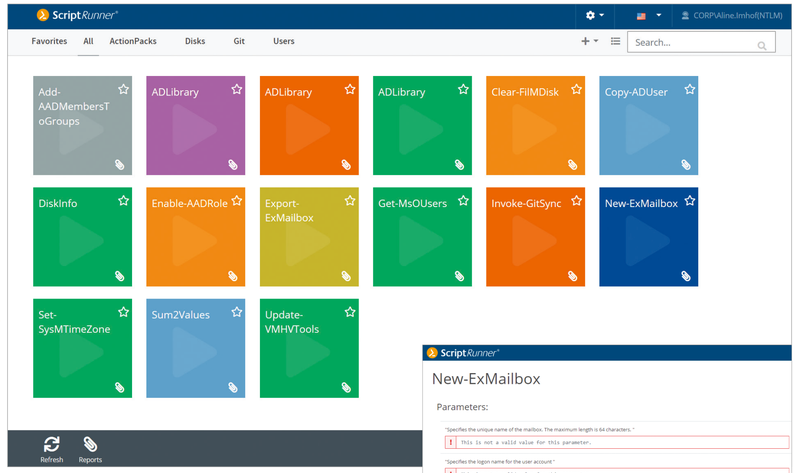
The Windows PowerShell ISE with ScriptRunner Plugin



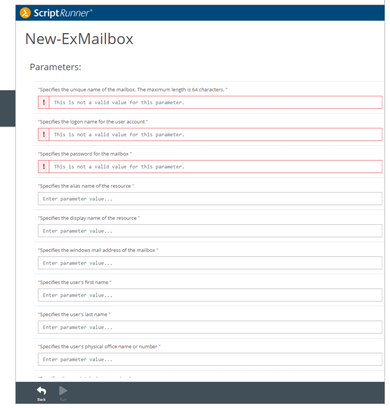
Action Configuration Wizard: Assign Parameter Values



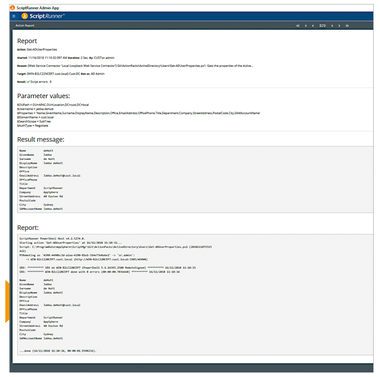
Delegation Configuration Wizard



Web based Delegation App



Automatically generated Web Form



Automatically generated PowerShell report

DELEGATION

- Securely delegate ScriptRunner **ACTION**s to groups, individual users, e.g., end users or service desk employees.
- When delegating actions, roles and rights are completely decoupled and ScriptRunner executes the action with the necessary administrative rights.

AUTOMATION

- Start ScriptRunner **ACTION**s from third-party systems such as monitoring, ITSM, or workflows. Connect with REST Web service and the email inbound connector.
- Connect with REST Web service and the email inbound connector.

REPORTS

- A **REPORT** contains all information about the execution of an **ACTION** and is split into several sections.
- The dashboard and the comparison functions allow a fast drill down and in-depth analysis of errors.