# About the Windows 7 Resource Kit PowerShell Pack

The Windows 7 Resource Kit PowersShell Pack is a collection of Windows PowerShell scripts included with the Resource Kit. The PowerShell Pack comes in the form of several Windows PowerShell modules, each containing anywhere from 3 to 600 functions.

To get started with the PowerShell pack, run InstallPowerShellPack.cmd from the directory on the CD, open up Windows PowerShell, and run Import-Module PowerShellPack[[1]](#endnote-1). After you run this command, you will have hundreds of PowerShell scripts loaded to play with.

The Windows 7 Resource Kit PowerShell Pack contains 10 modules to do all sorts of interesting things with PowerShell. Import-Module PowerShellPack actually imports 10 modules for you to use. Here’s a brief overview of each of the modules:

|  |  |
| --- | --- |
| **Module** | **Description** |
| WPK | Create rich user interfaces quick and easily from Windows PowerShell.  Think HTA, but easy.  Over 600 scripts to help you build quick user interfaces |
| TaskScheduler | List scheduled tasks, create or delete tasks |
| FileSystem | Monitor files and folders, check for duplicate files, and check disk space |
| IsePack | Supercharge your scripting in the Integrated Scripting Environment with over 35 shortcuts |
| DotNet | Explore loaded types, find commands that can work with a type, and explore how you can use PowerShell, DotNet and COM together. |
| PSImageTools | Convert, rotate, scale, and crop images and get image metadata |
| PSRSS | Harness the FeedStore  from PowerShell |
| PSSystemTools | Get Operating System or Hardware Information |
| PSUserTools | Get the users on a system, check for elevation, and start-processaadministrator |
| PSCodeGen | Generates PowerShell scripts, C# code, and PInvoke |
|  |  |

While you might not use all of these modules in every situation, they represent solid families of commands you can use in any PowerShell scrips you like. The examples below should also help you start to “think in PowerShell”. Most of the tasks you will do in Powershell will not use a single command, but will instead combine many commands in one or more pipelines to get the job done. Each step of these pipelines is a rich object, with properties, methods, and events. This set of commands gives you a number of tools for working with some rich objects that were already lying in the operating system, just waiting for you to discover.

### Exploring the Modules

To check out additional help for the modules, go ahead and try running:

Get-Help about\*module\*

This will show you all of the module topics, and you can just pick an individual module about topic to read. To get help on any of the commands from the PowerShell Pack, simply go ahead and do Get-Help *CommandName.*

## Individual Module Overview

### WPK

The WPF (Windows Presentation Foundation) PowerShell Kit allows you to build rich user interfaces entirely in PowerShell scripts. To get started, check out the next section: Writing User Interfaces with WPK.

## IsePack

The Windows PowerShell Integrated Scripting Environment is built into Powershell V2, and provides a nice way to edit and debug your scripts. The Integrated Scripting Environment can also be customized with shortcuts to accomplish common tasks.

IsePack (pronounced Ice Pack) is full of these shortcuts. If you load up IsePack, a menu called IsePack will be created beneath the AddOns menu in the Integrated Scripting Environment. Most of the items in IsePack have convenient shortcut keys. Here’s a few highlights:

|  |  |
| --- | --- |
| Add-InlineHelp | Quickly insert inline help into your functions so that Get-Help can help other users figure out how to use your code |
| Copy-Colored | Email scripts to your collegues in rich color |
| Copy-ColoredHTML | Blog out the scripts that you write with Copy-ColoredHTML |
| Show Syntax | Select a command and press CTRL + Y |
| Show Member | Save anything to a variable, and then select the variable and press ALT + M to see a grid view of the control’s members |

## FileSystem

You can use the filesystem module to check free disk space, create and add to zip files, watch locations on the filesystem, find duplicate files, or rename drives.

## DotNet

The DotNet module helps you work with the types loaded on the system. You can use Get-Type to search for loaded .NET types or you can use Get-ProgID to search for loaded COM types. For instance, try this:

Get-Type | Where-Object { $\_.Name –like “\*File\*” } | Select-Object FullName

To see all types related to files. You can try creating one of the types with New-Object or getting static members of the type with Get-Member –Static.

To look for COM types, like those used from VBScripts, use something like:  
Get-ProgID \*Image\*

## PSImageTools

The PSImageTools lets you manage photos of other images using Windows PowerShell. Resize, Rotate, or Crop images, or check out image metadata. Convert to JPEG or Bitmap. Here’s a quick example:

Get-ChildItem $env:UserProfile\Pictures | Get-Image |Get-ImageProperty

## PSRSS

PSRss lets you read your RSS feeds from Windows PowerShell. You can subscribe to new feeds, mark articles as read, and get feeds and descriptions. Here’s a quick pipeline to show the 10 most recent RSS articles.

Get-Feed |

Get-Article |

Sort-Object PubDate -Descending |

Select-Object Title, Description -First 10

## PSSystemTools

System Tools helps you get hardware and configuration information out of the operating system. Get information about USB devices, processors, boot status, fonts, and more. Check out this quick script to see the USB devices and their manufacturers.

Get-USB |

Select Name, Manufacturer

## PSUserTools

User Tools helps you deal with process elevation and users. You can test to see if the current user is an administrator, start processes that prompt for administrative credentials, get the users on the system, and get detailed information about the current user.

## PSCodeGen

PSCodeGen is a module to help advanced scripters create code more quickly by automatically generating the code. In PSCodeGen there is New-Enum, which allows you to define a new enumerated type, New-PInvoke, which allows you to work with the C APIs more easily, and New-ScriptCmdlet, which can be used to create new PowerShell advanced functions with ease. Check out these examples of using New-ScriptCmdlet to make new script cmdlets. The first example creates the Start-ProcessAsAdministrator script cmdlet that is in the PSUserTools module:

New-ScriptCmdlet -Name Start-ProcessAsAdministrator -FromCommand (Get-Command Start-Process) -RemoveParameter Verb -ProcessBlock {

$null = $psBoundParameters.Verb = "RunAs"

Start-Process @psBoundParameters

}

New-ScriptCmdlet -Name -FromCommand (Get-Command Get-Process) -RemoveParameter Verb

## TaskScheduler

The TaskScheduler module helps you use the Task Scheduler APIs available on Windows Vista and above to schedule running programs or scripts on your system. You can check running tasks, start tasks on demand. You can also start tasks with an incredible variety of triggers, like single time, daily, weekly, monthly, event log, workstation lock and workstation unlock . Here are some simple examples:

New-task |

Add-TaskTrigger -DayOfWeek Monday, Wednesday, Friday -WeeksInterval 2 -At "3:00 PM" |

Add-TaskAction -Script {

Get-Process | Out-GridView

Start-Sleep -Seconds 100

} |

Register-ScheduledTask TestTask

New-task |

Add-TaskTrigger -In (New-TimeSpan -Seconds 30) |

Add-TaskAction -Script {

Get-Process | Out-GridView

Start-Sleep -Seconds 100

} |

Register-ScheduledTask TestTask

1. If you have not changed your execution policy, you may get an error like this:

   cannot be loaded because the execution of scripts is disabled on this system. Please see "get-help about\_signing" for more details.

   In this case, you should consider changing your execution policy to RemoteSigned, Unrestricted, or Bypass. The Execution Policies are there for your protection, to help ensure that scripts you do not trust are not run by default. [↑](#endnote-ref-1)