MyWebUniversity

Full credit is given to the above companies including the OS that this PDF file was generated!

PowerShell Get-Help on command 'New-ModuleManifest'

PS C:\Users\wahid> Get-Help New-ModuleManifest

```
NAME
    New-ModuleManifest

SYNOPSIS
    Creates a new module manifest.


SYNTAX
    New-ModuleManifest [-Path] <System.String> [-AliasesToExport
    <System.String[]>] [-Author <System.String>] [-ClrVersion <System.Version>]
    [-CmdletsToExport <System.String[]>] [-CompanyName <System.String>]
    [-CompatiblePSEditions {Desktop | Core}] [-Copyright <System.String>]
    [-DefaultCommandPrefix <System.String>] [-Description <System.String>]
    [-DotNetFrameworkVersion <System.Version>] [-DscResourcesToExport
    <System.String[]>] [-FileList <System.String[]>] [-FormatsToProcess
    <System.String[]>] [-FunctionsToExport <System.String[]>] [-Guid
    <System.Guid>] [-HelpInfoUri <System.String>] [-IconUri <System.Uri>]
    [-LicenseUri <System.Uri>] [-ModuleList <System.Object[]>] [-ModuleVersion
    <System.Version>] [-NestedModules <System.Object[]>] [-PassThru]
    [-PowerShellHostName <System.String>] [-PowerShellHostVersion
    <System.Version>] [-PowerShellVersion <System.Version>] [-PrivateData
    <System.Object>] [-ProcessorArchitecture {None | MSIL | X86 | IA64 | Amd64 |
    Arm}] [-ProjectUri <System.Uri>] [-ReleaseNotes <System.String>]
    [-RequiredAssemblies <System.String[]>] [-RequiredModules <System.Object[]>]
    [-RootModule <System.String>] [-ScriptsToProcess <System.String[]>] [-Tags
    <System.String[]>] [-TypesToProcess <System.String[]>] [-VariablesToExport
    <System.String[]>] [-Confirm] [-WhatIf] [<CommonParameters>]
```

DESCRIPTION

The `New-ModuleManifest` cmdlet creates a new module manifest (`.psd1`) file, populates its values, and saves the manifest file in the specified path.

Module authors can use this cmdlet to create a manifest for their module. A module manifest is a `.psd1` file that contains a hash table. The keys and values in the hash table describe the contents and attributes of the module, define the prerequisites, and determine how the components are processed. Manifests aren't required for a module.

`New-ModuleManifest` creates a manifest that includes all the commonly used manifest keys, so you can use the default output as a manifest template. To add or change values, or to add module keys that this cmdlet doesn't add, open the resulting file in a text editor.

Each parameter, except for Path and PassThru , creates a module manifest key and its value. In a module manifest, only the ModuleVersion key is required. Unless specified in the parameter description, if you omit a parameter from the command, `New-ModuleManifest` creates a comment string for the associated value that has no effect.

In PowerShell 2.0, `New-ModuleManifest` prompts you for the values of commonly used parameters that aren't specified in the command, in addition to required parameter values. Beginning in PowerShell 3.0, `New-ModuleManifest` prompts only when required parameter values aren't specified.

If you are planning to publish your module in the PowerShell Gallery, the
manifest must contain values for certain properties. For more information, see
Required metadata for items published to the PowerShell Gallery (/powershell/ga
llery/how-to/publishing-packages/publishing-a-package#required-metadata-for-ite
ms-published-to-the-powershell-gallery)in the Gallery documentation.

PARAMETERS

-AliasesToExport <System.String[]>

Specifies the aliases that the module exports. Wildcards are permitted.

You can use this parameter to restrict the aliases that are exported by
the module. It can remove aliases from the list of exported aliases, but
it can't add aliases to the list.

If you omit this parameter, `New-ModuleManifest` creates an
AliasesToExport key with a value of `*` (all), meaning that all aliases
defined in the module are exported by the manifest.

-Author <System.String>

Specifies the module author.

If you omit this parameter, `New-ModuleManifest` creates an Author key
with the name of the current user.

-ClrVersion <System.Version>

Specifies the minimum version of the Common Language Runtime (CLR) of the
Microsoft .NET Framework that the module requires.

> [!NOTE] > This setting is valid for the PowerShell Desktop edition only,
such as Windows PowerShell 5.1, > and only applies to .NET Framework
versions lower than 4.5. This requirement has no effect for > newer

versions of PowerShell or the .NET Framework.

-CmdletsToExport <System.String[]>

Specifies the cmdlets that the module exports. Wildcards are permitted.

You can use this parameter to restrict the cmdlets that are exported by the module. It can remove cmdlets from the list of exported cmdlets, but it can't add cmdlets to the list.

If you omit this parameter, `New-ModuleManifest` creates a CmdletsToExport key with a value of `*` (all), meaning that all cmdlets defined in the module are exported by the manifest.

-CompanyName <System.String>

Identifies the company or vendor who created the module.

If you omit this parameter, `New-ModuleManifest` creates a CompanyName key with a value of "Unknown".

-CompatiblePSEditions <System.String[]>

Specifies the module's compatible PSEditions. For information about PSEdition, see Modules with compatible PowerShell Editions (/powershell/gallery/concepts/module-psedition-support).

-Copyright <System.String>

Specifies a copyright statement for the module.

If you omit this parameter, `New-ModuleManifest` creates a Copyright key with a value of `(c) <year> <username>. All rights reserved.` where `<year>` is the current year and `<username>` is the value of the Author key.

-DefaultCommandPrefix <System.String>

Specifies a prefix that is prepended to the nouns of all commands in the module when they're imported into a session. Enter a prefix string. Prefixes prevent command name conflicts in a user's session.

Module users can override this prefix by specifying the Prefix parameter of the `Import-Module` cmdlet.

This parameter was introduced in PowerShell 3.0.

-Description <System.String>

Describes the contents of the module.

-DotNetFrameworkVersion <System.Version>

Specifies the minimum version of the Microsoft .NET Framework that the module requires.

> [!NOTE] > This setting is valid for the PowerShell Desktop edition only, such as Windows PowerShell 5.1, > and only applies to .NET Framework versions lower than 4.5. This requirement has no effect for > newer versions of PowerShell or the .NET Framework.

-DscResourcesToExport <System.String[]>

Specifies the Desired State Configuration (DSC) resources that the module exports. Wildcards are permitted.

-FileList <System.String[]>

Specifies all items that are included in the module.

This key is designed to act as a module inventory. The files listed in the key are included when the module is published, but any functions aren't automatically exported.

-FormatsToProcess <System.String[]>

Specifies the formatting files (`.ps1xml`) that run when the module is
imported.

When you import a module, PowerShell runs the `Update-FormatData` cmdlet
with the specified files. Because formatting files aren't scoped, they
affect all session states in the session.

-FunctionsToExport <System.String[]>

Specifies the functions that the module exports. Wildcards are permitted.

You can use this parameter to restrict the functions that are exported by
the module. It can remove functions from the list of exported aliases, but
it can't add functions to the list.

If you omit this parameter, `New-ModuleManifest` creates an
FunctionsToExport key with a value of `*` (all), meaning that all
functions defined in the module are exported by the manifest.

-Guid <System.Guid>

Specifies a unique identifier for the module. The GUID can be used to
distinguish among modules with the same name.

If you omit this parameter, `New-ModuleManifest` creates a GUID key in the
manifest and generates a GUID for the value.

To create a new GUID in PowerShell, type `[guid]::NewGuid()`.

-HelpInfoUri <System.String>

Specifies the internet address of the HelpInfo XML file for the module.
Enter a Uniform Resource Identifier (URI) that begins with http or https .

The HelpInfo XML file supports the Updatable Help feature that was
introduced in PowerShell 3.0. It contains information about the location

of downloadable help files for the module and the version numbers of the newest help files for each supported locale.

For information about Updatable Help, see about_Updatable_Help (./About/about_Updatable_Help.md). For information about the HelpInfo XML file, see Supporting Updatable Help (/powershell/scripting/developer/module/supporting-updatable-help).

This parameter was introduced in PowerShell 3.0.

-IconUri <System.Uri>

Specifies the URL of an icon for the module. The specified icon is displayed on the gallery web page for the module.

-LicenseUri <System.Uri>

Specifies the URL of licensing terms for the module.

-ModuleList <System.Object[]>

Lists all modules that are included in this module.

Enter each module name as a string or as a hash table with ModuleName and ModuleVersion keys. The hash table can also have an optional GUID key. You can combine strings and hash tables in the parameter value.

This key is designed to act as a module inventory. The modules that are listed in the value of this key aren't automatically processed.

-ModuleVersion <System.Version>

Specifies the module's version.

This parameter isn't required, but a ModuleVersion key is required in the manifest. If you omit this parameter, `New-ModuleManifest` creates a ModuleVersion key with a value of 1.0.

-NestedModules <System.Object[]>

Specifies script modules (`.psm1`) and binary modules (`.dll`) that are imported into the module's session state. The files in the NestedModules key run in the order in which they're listed in the value.

Enter each module name as a string or as a hash table with ModuleName and ModuleVersion keys. The hash table can also have an optional GUID key. You can combine strings and hash tables in the parameter value.

Typically, nested modules contain commands that the root module needs for its internal processing. By default, the commands in nested modules are exported from the module's session state into the caller's session state, but the root module can restrict the commands that it exports. For example, by using an `Export-ModuleMember` command.

Nested modules in the module session state are available to the root module, but they aren't returned by a `Get-Module` command in the caller's session state.

Scripts (`.ps1`) that are listed in the NestedModules key are run in the module's session state, not in the caller's session state. To run a script in the caller's session state, list the script file name in the value of the ScriptsToProcess key in the manifest.

-PassThru <System.Management.Automation.SwitchParameter>

Writes the resulting module manifest to the console and creates a `.psd1` file. By default, this cmdlet doesn't generate any output.

-Path <System.String>

Specifies the path and file name of the new module manifest. Enter a path and file name with a `.psd1` file name extension, such as `$pshome\Modules\MyModule\MyModule.psd1`. The Path parameter is required.

If you specify the path to an existing file, `New-ModuleManifest` replaces the file without warning unless the file has the read-only attribute.

The manifest should be located in the module's directory, and the manifest file name should be the same as the module directory name, but with a `.psd1` file name extension.

> [!NOTE] > You cannot use variables, such as `$PSHOME` or `$HOME`, in response to a prompt for a Path > parameter value. To use a variable, include the Path parameter in the command.

-PowerShellHostName <System.String>

Specifies the name of the PowerShell host program that the module requires. Enter the name of the host program, such as Windows PowerShell ISE Host or ConsoleHost . Wildcards aren't permitted.

To find the name of a host program, in the program, type `$Host.Name`.

-PowerShellHostVersion <System.Version>

Specifies the minimum version of the PowerShell host program that works with the module. Enter a version number, such as 1.1.

-PowerShellVersion <System.Version>

Specifies the minimum version of PowerShell that works with this module. For example, you can enter 1.0, 2.0, or 3.0 as the parameter's value. It must be in an X.X format. For example, if you submit `5`, PowerShell will throw an error.

-PrivateData <System.Object>

Specifies data that is passed to the module when it's imported.

-ProcessorArchitecture <System.Reflection.ProcessorArchitecture>

Specifies the processor architecture that the module requires. Valid
values are x86, AMD64, IA64, MSIL, and None (unknown or unspecified).

-ProjectUri <System.Uri>

Specifies the URL of a web page about this project.

-ReleaseNotes <System.String>

Specifies release notes.

-RequiredAssemblies <System.String[]>

Specifies the assembly (`.dll`) files that the module requires. Enter the assembly file names. PowerShell loads the specified assemblies before updating types or formats, importing nested modules, or importing the module file that is specified in the value of the RootModule key.

Use this parameter to list all the assemblies that the module requires, including assemblies that must be loaded to update any formatting or type files that are listed in the FormatsToProcess or TypesToProcess keys, even if those assemblies are also listed as binary modules in the NestedModules key.

-RequiredModules <System.Object[]>

Specifies modules that must be in the global session state. If the required modules aren't in the global session state, PowerShell imports them. If the required modules aren't available, the `Import-Module` command fails.

Enter each module name as a string or as a hash table with ModuleName and ModuleVersion keys. The hash table can also have an optional GUID key. You can combine strings and hash tables in the parameter value.

In PowerShell 2.0, `Import-Module` doesn't import required modules automatically. It just verifies that the required modules are in the global session state.

-RootModule <System.String>

Specifies the primary or root file of the module. Enter the file name of a script (`.ps1`), a script module (`.psm1`), a module manifest(`.psd1`), an assembly (`.dll`), a cmdlet definition XML file (`.cdxml`), or a workflow (`.xaml`). When the module is imported, the members that are exported from the root module file are imported into the caller's session state.

If a module has a manifest file and no root file was designated in the RootModule key, the manifest becomes the primary file for the module, and the module becomes a manifest module (ModuleType = Manifest).

To export members from `.psm1` or `.dll` files in a module that has a manifest, the names of those files must be specified in the values of the RootModule or NestedModules keys in the manifest. Otherwise, their members aren't exported.

> [!NOTE] > In PowerShell 2.0, this key was called ModuleToProcess . You can use the RootModule > parameter name or its ModuleToProcess alias.

-ScriptsToProcess <System.String[]>

Specifies script (`.ps1`) files that run in the caller's session state when the module is imported. You can use these scripts to prepare an environment, just as you might use a login script.

To specify scripts that run in the module's session state, use the NestedModules key.

-Tags <System.String[]>

Specifies an array of tags.

-TypesToProcess <System.String[]>

    Specifies the type files (`.ps1xml`) that run when the module is imported.

    When you import the module, PowerShell runs the `Update-TypeData` cmdlet
    with the specified files. Because type files aren't scoped, they affect
    all session states in the session.

-VariablesToExport <System.String[]>
    Specifies the variables that the module exports. Wildcards are permitted.

    You can use this parameter to restrict the variables that are exported by
    the module. It can remove variables from the list of exported variables,
    but it can't add variables to the list.

    If you omit this parameter, `New-ModuleManifest` creates a
    VariablesToExport key with a value of `*` (all), meaning that all
    variables defined in the module are exported by the manifest.

-Confirm <System.Management.Automation.SwitchParameter>
    Prompts you for confirmation before running the cmdlet.

-WhatIf <System.Management.Automation.SwitchParameter>
    Shows what would happen if `New-ModuleManifest` runs. The cmdlet isn't run.

<CommonParameters>
    This cmdlet supports the common parameters: Verbose, Debug,
    ErrorAction, ErrorVariable, WarningAction, WarningVariable,
    OutBuffer, PipelineVariable, and OutVariable. For more information, see
    about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

----------- Example 1 - Create a new module manifest -----------

```
New-ModuleManifest -Path C:\ps-test\Test-Module\Test-Module.psd1 -PassThru
```

```
#
# Module manifest for module 'Test-Module'
#
# Generated by: ContosoAdmin
#
# Generated on: 1/22/2019
#

@{

# Script module or binary module file associated with this manifest.
# RootModule = ''

# Version number of this module.
ModuleVersion = '1.0'

# Supported PSEditions
# CompatiblePSEditions = @()

# ID used to uniquely identify this module
GUID = '47179120-0bcb-4f14-8d80-f4560107f85c'

# Author of this module
Author = 'ContosoAdmin'

# Company or vendor of this module
CompanyName = 'Unknown'

# Copyright statement for this module
Copyright = '(c) 2019 ContosoAdmin. All rights reserved.'

# Description of the functionality provided by this module
# Description = ''
# Minimum version of the Windows PowerShell engine required by this module
# PowerShellVersion = ''

# Name of the Windows PowerShell host required by this module
# PowerShellHostName = ''

# Minimum version of the Windows PowerShell host required by this module
# PowerShellHostVersion = ''

# Minimum version of Microsoft .NET Framework required by this module. This
prerequisite is valid for the PowerShell Desktop edition only.
# DotNetFrameworkVersion = ''

# Minimum version of the common language runtime (CLR) required by this
module. This prerequisite is valid for the PowerShell Desktop edition only.
# CLRVersion = ''

# Processor architecture (None, X86, Amd64) required by this module
# ProcessorArchitecture = ''

# Modules that must be imported into the global environment prior to importing
this module
# RequiredModules = @()

# Assemblies that must be loaded prior to importing this module
# RequiredAssemblies = @()

# Script files (.ps1) that are run in the caller's environment prior to
importing this module.
# ScriptsToProcess = @()

# Type files (.ps1xml) to be loaded when importing this module
# TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module
# FormatsToProcess = @()

# Modules to import as nested modules of the module specified in
RootModule/ModuleToProcess
# NestedModules = @()

# Functions to export from this module, for best performance, do not use
wildcards and do not delete the entry, use an empty array if there are no
functions to export.
FunctionsToExport = @()

# Cmdlets to export from this module, for best performance, do not use
wildcards and do not delete the entry, use an empty array if there are no
cmdlets to export.
CmdletsToExport = @()

# Variables to export from this module
VariablesToExport = '*'

# Aliases to export from this module, for best performance, do not use
wildcards and do not delete the entry, use an empty array if there are no
aliases to export.
AliasesToExport = @()

# DSC resources to export from this module
# DscResourcesToExport = @()

# List of all modules packaged with this module
# ModuleList = @()
# List of all files packaged with this module
# FileList = @()

# Private data to pass to the module specified in RootModule/ModuleToProcess.
This may also contain a PSData hashtable with additional module metadata used
by PowerShell.
PrivateData = @{

    PSData = @{

        # Tags applied to this module. These help with module discovery in
online galleries.
        # Tags = @()

        # A URL to the license for this module.
        # LicenseUri = ''

        # A URL to the main website for this project.
        # ProjectUri = ''

        # A URL to an icon representing this module.
        # IconUri = ''

        # ReleaseNotes of this module
        # ReleaseNotes = ''

    } # End of PSData hashtable

} # End of PrivateData hashtable

# HelpInfo URI of this module
# HelpInfoURI = ''
# Default prefix for commands exported from this module. Override the default
prefix using Import-Module -Prefix.
# DefaultCommandPrefix = ''

}
```

Example 2 - Create a new manifest with some prepopulated settings

```
$moduleSettings = @{
    PowerShellVersion = 1.0
    Path   = 'C:\ps-test\ManifestTest.psd1'
    AliasesToExport   = @(
      'JKBC'
      'DRC'
      'TAC'
    )
}
New-ModuleManifest @moduleSettings
```

-- Example 3 - Create a manifest that requires other modules --

```
$moduleSettings = @{
  RequiredModules = ("BitsTransfer", @{
    ModuleName="PSScheduledJob"
    ModuleVersion="1.0.0.0";
    GUID="50cdb55f-5ab7-489f-9e94-4ec21ff51e59"
  })
  Path = 'C:\ps-test\ManifestTest.psd1'
}
New-ModuleManifest @moduleSettings
```

This example shows how to use the string and hash table formats of the ModuleList , RequiredModules , and NestedModules parameter. You can combine strings and hash tables in the same parameter value.

- Example 4 - Create a manifest that supports updateable help -

```
$moduleSettings = @{
  HelpInfoUri = 'http://https://go.microsoft.com/fwlink/?LinkID=603'
  Path = 'C:\ps-test\ManifestTest.psd1'
}
New-ModuleManifest @moduleSettings
```

For information about Updatable Help, see about_Updatable_Help (./About/about_Updatable_Help.md). For information about the HelpInfo XML file, see Supporting Updatable Help (/powershell/scripting/developer/module/supporting-updatable-help).

------------ Example 5 - Getting module information ------------

```
Get-Module Microsoft.PowerShell.Diagnostics -List | Format-List -Property *
```

```
LogPipelineExecutionDetails : False
Name                        : Microsoft.PowerShell.Diagnostics
Path                        : C:\Windows\system32\WindowsPowerShell\v1.0\Module
s\Microsoft.PowerShell.Diagnostics\Micro
                              soft.PowerShell.Diagnostics.psd1
Definition                  :
Description                 :
Guid                        : ca046f10-ca64-4740-8ff9-2565dba61a4f
HelpInfoUri                 : https://go.microsoft.com/fwlink/?LinkID=210596
ModuleBase                  : C:\Windows\system32\WindowsPowerShell\v1.0\Module
s\Microsoft.PowerShell.Diagnostics
PrivateData                 :
Version                     : 3.0.0.0
ModuleType                  : Manifest
```

```
Author                    : Microsoft Corporation
AccessMode                : ReadWrite
ClrVersion                : 4.0
CompanyName               : Microsoft Corporation
Copyright                 : Microsoft Corporation. All rights reserved.
DotNetFrameworkVersion    :
ExportedFunctions         : {}
ExportedCmdlets           : {[Get-WinEvent, Get-WinEvent], [Get-Counter,
Get-Counter], [Import-Counter,
                            Import-Counter], [Export-Counter,
Export-Counter]...}
ExportedCommands          : {[Get-WinEvent, Get-WinEvent], [Get-Counter,
Get-Counter], [Import-Counter,
                            Import-Counter], [Export-Counter,
Export-Counter]...}
FileList                  : {}
ModuleList                : {}
NestedModules             : {}
PowerShellHostName        :
PowerShellHostVersion     :
PowerShellVersion         : 3.0
ProcessorArchitecture     : None
Scripts                   : {}
RequiredAssemblies        : {}
RequiredModules           : {}
RootModule                :
ExportedVariables         : {}
ExportedAliases           : {}
ExportedWorkflows         : {}
SessionState              :
OnRemove                  :
ExportedFormatFiles       :
{C:\Windows\system32\WindowsPowerShell\v1.0\Event.format.ps1xml,
    C:\Windows\system32\WindowsPowerShell\v1.0\Diagnostics.format.ps1xml}
    ExportedTypeFiles        :
    {C:\Windows\system32\WindowsPowerShell\v1.0\GetEvent.types.ps1xml}



REMARKS
    To see the examples, type: "get-help New-ModuleManifest -examples".
    For more information, type: "get-help New-ModuleManifest -detailed".
    For technical information, type: "get-help New-ModuleManifest -full".
    For online help, type: "get-help New-ModuleManifest -online"
```