Windows PowerShell Get-Help on Cmdlet 'Get-EventSubscriber'

PS:\>Get-HELP Get-EventSubscriber -Full

NAME

Get-EventSubscriber

SYNOPSIS

Gets the event subscribers in the current session.

SYNTAX

Get-EventSubscriber [[-SourceIdentifier] <System.String>] [[-Force]] [<CommonParameters>]

Get-EventSubscriber [-SubscriptionId] <System.Int32> [[-Force]] [<CommonParameters>]

DESCRIPTION

The `Get-EventSubscriber` cmdlet gets the event subscribers in the current session.

When you subscribe to an event by using a Register event cmdlet, an event subscriber is added to your Windows PowerShell session, and the events to which you

subscribed are added to your event queue whenever they are raised. To cancel an event subscription, delete the event subscriber by using the `Unregister-Event` cmdlet.

PARAMETERS

-Force <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet gets all event subscribers, including subscribers for events that are hidden by using the SupportEvent parameter of

`Register-ObjectEvent`, `Register-WmiEvent`, and `Register-EngineEvent`.

Required? false

Position? 1

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-SourceIdentifier <System.String>

Specifies the SourceIdentifier property value that gets only the event subscribers. By default, `Get-EventSubscriber` gets all event subscribers in the session.

Wildcards are not permitted. This parameter is case-sensitive.

Required? false

Position? 0

Default value None

Accept pipeline input? True (ByPropertyName)

Accept wildcard characters? false

-SubscriptionId <System.Int32>

Specifies the subscription identifier that this cmdlet gets. By default, `Get-EventSubscriber` gets all event subscribers in the session.

Required? true

Position? 0

Default value None

Accept pipeline input? True (ByPropertyName)

Accept wildcard characters? false

<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).

INPUTS

None

You can't pipe objects to this cmdlet.

OUTPUTS

System.Management.Automation.PSEventSubscriber

This cmdlet returns a PSEventSubscriber object for each event subscriber.

NOTES

The `New-Event` cmdlet, which creates a custom event, does not generate a subscriber. Therefore, the `Get-EventSubscriber` cmdlet will not find a subscriber

object for these events. However, if you use the `Register-EngineEvent` cmdlet to subscribe to a custom event (in order to forward the event or to specify an

action), `Get-EventSubscriber` will find the subscriber that `Register-EngineEvent` generates.

Events, event subscriptions, and the event queue exist only in the current session. If you close the current session, the event queue is discarded and the event

subscription is canceled.

---- Example 1: Get the event subscriber for a timer event ----

```
$Timer = New-Object Timers.Timer
$Timer | Get-Member -Type Event

TypeName: System.Timers.Timer

Name     MemberType Definition
----     ---------- ----------
Disposed Event      System.EventHandler Disposed(System.Object, System.EventArgs)
Elapsed  Event      System.Timers.ElapsedEventHandler Elapsed(System.Object, System.Timers.ElapsedEventArgs)

Register-ObjectEvent -InputObject $Timer -EventName Elapsed -SourceIdentifier Timer.Elapsed
Get-EventSubscriber

SubscriptionId   : 4
SourceObject     : System.Timers.Timer
EventName        : Elapsed
SourceIdentifier : Timer.Elapsed
Action           :
HandlerDelegate  :
SupportEvent     : False
ForwardEvent     : False
```

The third command uses the `Register-ObjectEvent` cmdlet to register for the Elapsed event on the timer object.

The fourth command uses the `Get-EventSubscriber` cmdlet to get the event subscriber for the Elapsed event.

Example 2: Use the dynamic module in PSEventJob in the Action property of the event subscriber

```
$Timer = New-Object Timers.Timer
$Timer.Interval = 500
$params = @{
    InputObject = $Timer
    EventName = 'Elapsed'
    SourceIdentifier = 'Timer.Random'
```

```
    Action = { $Random = Get-Random -Min 0 -Max 100 }
}
Register-ObjectEvent @params

Id  Name          State      HasMoreData  Location  Command
--  ----          -----      -----------  --------  -------
3   Timer.Random  NotStarted False                 $Random = Get-Random ...

$Timer.Enabled = $True
$Subscriber = Get-EventSubscriber -SourceIdentifier Timer.Random
($Subscriber.action).gettype().fullname

System.Management.Automation.PSEventJob

$Subscriber.action | Format-List -Property *

State         : Running
Module        : __DynamicModule_6b5cbe82-d634-41d1-ae5e-ad7fe8d57fe0
StatusMessage :
HasMoreData   : True
Location      :
Command       : $random = Get-Random -Min 0 -Max 100
JobStateInfo  : Running
Finished      : System.Threading.ManualResetEvent
InstanceId    : 88944290-133d-4b44-8752-f901bd8012e2
Id            : 1
Name          : Timer.Random
ChildJobs     : {}
...

& $Subscriber.action.module {$Random}
```

The third command uses the `Register-ObjectEvent` cmdlet to register the Elapsed event of the timer object. The

command includes an action that handles the event.

Whenever the timer interval elapses, an event is raised and the commands in the action run. In this case, the `Get-Random` cmdlet generates a random number between 0

and 100 and saves it in the `$Random` variable. The source identifier of the event is Timer.Random.

When you use an Action parameter in a `Register-ObjectEvent` command, the command returns a PSEventJob object that represents the action.

The fourth command enables the timer.

The fifth command uses the `Get-EventSubscriber` cmdlet to get the event subscriber of the Timer.Random event. It saves the event subscriber object in the

`$Subscriber` variable.

The sixth command shows that the Action property of the event subscriber object contains a PSEventJob object. In fact, it contains the same PSEventJob object that the

`Register-ObjectEvent` command returned.

The seventh command uses the `Format-List` cmdlet to display all of the properties of the PSEventJob object in the Action property in a list. The result reveals that

the PSEventJob object has a Module property that contains a dynamic script module that implements the action.

The remaining commands use the call operator (`&`) to invoke the command in the module and display the value of the $Random variable. You can use the call operator to

invoke any command in a module, including commands that are not exported. In this case, the commands show the random number that is being generated when the Elapsed

event occurs.

For more information about modules, see about_Modules (../Microsoft.PowerShell.Core/About/about_Modules.md).

RELATED LINKS

Online Version:

https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/get-eventsubscriber?view=powershell-5.1&WT.mc

_id=ps-gethelp

Get-Event

New-Event

Register-EngineEvent

Register-ObjectEvent

Remove-Event

Unregister-Event

Wait-Event