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

PowerShell Get-Help on command 'Convert-String'

PS C:\Users\wahid> Get-Help Convert-String

NAME

Convert-String

SYNOPSIS

Formats a string to match examples.

SYNTAX

Convert-String [-Example

<System.Collections.Generic.List`1[System.Management.Automation.PSObject]>]

-InputObject <System.String> [<CommonParameters>]

DESCRIPTION

The cmdlet formats a string to match the format of examples.

PARAMETERS

-Example

<System.Collections.Generic.List`1[System.Management.Automation.PSObject]>

Specifies a list of examples of the target format. Specify pairs separated

by the equal sign (`=`), with the source pattern on the left and the
target pattern on the right, as in the following examples:

- `-Example "Hello World=World, Hello"`

- `-Example "Hello World=World: Hello",'"Hello","1"=1: Hello'`

> [!NOTE] > The second example uses a list of patterns

Alternatively, specify a list of hash tables that contain Before and After
properties.

- `-Example @{before='"Hello","World"'; after='World: Hello'},
@{before='"Hello","1"'; after='1: Hello'}`
> [!CAUTION] > Avoid using spaces around the equal sign(`=`), as they are
treated as part of the pattern.

-InputObject <System.String>
Specifies a string to format.

<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: Convert format of a string ----------

"Mu Han", "Jim Hance", "David Ahs", "Kim Akers" | Convert-String -Example "Ed
Wilson=Wilson, E."

Han, M.

Hance, J.

Ahs, D.

Akers, K.

The first command creates an array that contains first and last names.

The second command formats the names according to the example. It puts the surname first in the output, followed by an initial.

---------- Example 2: Simplify format of a string ----------

$composers = @("Johann Sebastian Bach", "Wolfgang Amadeus Mozart", "Frederic Francois Chopin", "Johannes Brahms")

$composers | Convert-String -Example "first middle last=last, first"

Bach, Johann

Mozart, Wolfgang

Chopin, Frederic

Brahms, Johannes

The first command creates an array that contains first, middle and last names. Note that the last entry has no middle name.

The second command formats the names according to the example. It puts the last name first in the output, followed by the first name. All middle names removed; entry without middle name is handled correctly.

Example 3: Output management when strings don't match example

$composers = @("Johann Sebastian Bach", "Wolfgang Amadeus Mozart", "Frederic Francois Chopin", "Johannes Brahms")

$composers | Convert-String -Example "first middle last=middle, first"

Sebastian, Johann

Amadeus, Wolfgang

Francois, Frederic

The first command creates an array that contains first, middle and last names. Note that the last entry has no middle name.

The second command formats the names according to the example. It puts the middle name first in the output, followed by the first name. The last entry in `$Composers` is skipped, because it doesn't match the sample pattern: it has no middle name.

---------- Example 4: Caution with beauty spaces ----------

```
$composers = @("Antonio Vivaldi", "Richard Wagner ", "Franz Schubert",
"Johannes Brahms ")
$composers | Convert-String -Example "Patti Fuller = Fuller, P."
```

Wagner, R.

Brahms, J.

The first command creates an array of first and last names. Note that second and fourth items have an extra trailing space, after the last name.

The second command converts all strings that match the sample pattern: word, space, word, and final trailing space , all of this before the equal sign(`=`). Also, note the leading space in the output.

- Example 5: Format process information with multiple patterns -

```
$ExamplePatterns = @(
    @{before='"Hello","World"'; after='World: Hello'},
    @{before='"Hello","1"'; after='1: Hello'},
    @{before='"Hello-World","22"'; after='22: Hello-World'},
    @{before='"hello world","333"'; after='333: hello world'}
)
```

```
$Processes = Get-Process   | Select-Object -Property ProcessName, Id |
ConvertTo-Csv -NoTypeInformation
$Processes | Convert-String -Example $ExamplePatterns

Id: ProcessName
4368: AGSService
8896: Amazon Music Helper
4420: AppleMobileDeviceService
...
11140: git-bash
0: Idle
...
56: Secure System
...
13028: WmiPrvSE
2724: WUDFHost
2980: WUDFHost
3348: WUDFHost
```

$ExamplePatterns defines different expected patterns in the data, through examples.

The first pattern, `@{before='"Hello","World"'; after='World: Hello'}`, reads as follows:

- expect strings where a word comes enclosed in double quotes, then a comma, - and then the second, and last, word enclosed in quotes; - with no spaces in the string. On the output: place second word first, - without quotes, then a single space, and then the first word, without quotes. The second pattern, `@{before='"Hello","1"'; after='1: Hello'}`, reads as follows:

- expect strings where a word comes enclosed in double quotes, then a comma, - and then a number enclosed in quotes; - with no spaces in the string. On the

output: place the number first, - without quotes, then a single space, and then the word, without quotes. The third pattern, `@{before='"Hello-World","22"'; after='22: Hello-World'}`, reads as follows:

- expect strings where two words with a hyphen in between come enclosed in - double quotes, then a comma, and then a number enclosed in quotes; - with no spaces between the comma and the third double quote. - On the output: place the number first, without quotes, then a single space, - and then the hyphenated words, without quotes. The fourth, and final, pattern, `@{before='"hello world","333"'; after='333: hello world'}`, reads as follows:

- expect strings where two words with a space in between come enclosed in - double quotes, then a comma, and then a number enclosed in quotes; - with no spaces between the comma and the third double quote. - On the output: place the number first, without quotes, then a single space, - and then the words with the space in between, without quotes. The first command gets all processes by using the Get-Process cmdlet. The command passes them to the Select-Object cmdlet, which selects the process name and process ID. At the end of the pipeline, the command converts the output to comma separated values, without type information, by using the ConvertTo-Csv cmdlet. The command stores the results in the $Processes variable. $Processes now contains process names and PID.

The second command specifies an example variable that changes the order of the input items. The command coverts each string in `$Processes`.

> [!NOTE] > The fourth pattern implicitly says that two or more words separated by spaces are matched. Without > the fourth pattern, only the first word of the string enclosed in double quotes is matched.

REMARKS

To see the examples, type: "get-help Convert-String -examples".
For more information, type: "get-help Convert-String -detailed".
For technical information, type: "get-help Convert-String -full".

For online help, type: "get-help Convert-String -online"