**Microcode: PowerShell Scripting Tricks: The Joy of using Hashtables with Windows PowerShell**

[JamesBrundage](http://blogs.msdn.com/144972/ProfileUrlRedirect.ashx)

27 Nov 2008 2:01 AM

* Comments [4](http://blogs.msdn.com/b/mediaandmicrocode/archive/2008/11/27/microcode-powershell-scripting-tricks-the-joy-of-using-hashtables-with-windows-powershell.aspx#comments)

PowerShell is full of nice little touches.  While some of these touches are easy to run across, like Get-Command and Get-Help, there are also a lot of little ones that are very small changes that make working with some parts of PowerShell (and .NET) a whole lot easier.  One area that's got a lot of little goodness, but that probably isn't as well-known, is the joy of using a Hashtable in PowerShell.

For the uninitiated, you can create an instance of System.Collections.Hashtable in Windows PowerShell by the simple shorthand @{Key=Value}.  This offers a way you can express rather complicated data in a PowerShell script while keeping clean looking scripts.  For instance, here's how some moderately complex personal information might look as a hashtable.

$person = @{

First = "James"

Last = "Brundage"

Address = @{

City = "Seattle"

State = "Washington"

Zip = 98102

Street = "0 Way I Am Putting This On a Blog"

}

Occupation = @{

Industry = "Information Technology"

Title="Software Developer In Test"

Company="Microsoft Corporation"

}

}

Now that we've covered the basics, lets get to the cool tricks.

**Did you know? in PowerShell + adds Hashtables together.**

Watch how you use this on $person to make a new key:

$person+=@{"FullName" = "$($person.First) $($person.Last)"}

Nifty, right?

**Did you know?  You can use the . operator to get or set a Hashtable entry.**

$person.FullName

**Since you can peek at values with a ., you can do a quick check to fill in information that might be missing from the hashtable.**

Check out how to fill in the missing information from John Doe

$otherPerson = @{

First = "John"

Last = "Doe"

}

$people = $person, $otherPerson

$people = $people | Foreach-Object {

$p = $\_ + @{"FullName" = "$($\_.First) $($\_.Last)"}

if (-not $p.Address) {

$p.Address = "Unknown"

}

if (-not $p.Occupation) {

$p.Occupation = "Unknown"

}

$p

}

$people

**You can also access nested hashtables this way, which lets you group complex data.**

Watch how to use this to create a new variable, $westCoastPeople:

$westCoastPeople = $people |

Where-Object {"Washington", "Alaska", "Hawaii", "Oregon", "California" -contains $\_.Address.State }

**You can also use Group-Object to group by different properties.**

$people | Group-Object {$\_.Address.City, $\_.Address.State}

**If you need to walk through each item in a hashtable, then you can use .GetEnumerator() to walk the entries.**

$person.GetEnumerator() | Foreach-Object {

$\_.Key

"-"

$\_.Value | Out-String

}

You can't remove an item while you're enumerating it, but by wrapping the enumerator in @(), you actually get a snapshot of collection and then you're free to modify it as you see fit.

Check out how you can use this to clear out any field that has "Unknown" in it.

$people | Foreach-Object {

$p = $\_

@($p.GetEnumerator()) | Where-Object {

($\_.Value | Out-String) -like "\*Unknown\*"

} | Foreach-Object {

$p.Remove($\_.Key)

}

}

$people

As you can see, several nooks and crannies of how PowerShell works with .NET are quite useful, and make working with some objects in PowerShell incredibly nice.  I hope you've had some fun with hashtables, and I hope I can shed some more light on fun with different .NET types in the future.

Hope this helps,

James Brundage [MSFT]