Microsoft | TechNet

Customizing the Windows PowerShell Console

This is your guide to getting started with Windows PowerShell. Read through these pages to get familiar with Windows PowerShell, and soon you'll be driving around like a pro.

On This Page

Customizing the Console

Many of you will find this hard to believe, but there was a time - long, long ago - when people would get things and then use those items *exactly as they got them*! That's right: no customization, no "tricking out," no modding, no skinning, no nothing. You just took the thing out of the box and used it the way nature - and the manufacturer - intended.

Yes, we know: barbaric.

Today, of course, things are very different. No one would ever *dream* of using an off-the-shelf item in this day and age; instead, everything needs to be personalized. And that might have some of you a bit leery about trying Windows PowerShell. After all, if you can't customize and trick out this new software, well, then what's the point in even using it in the first place?

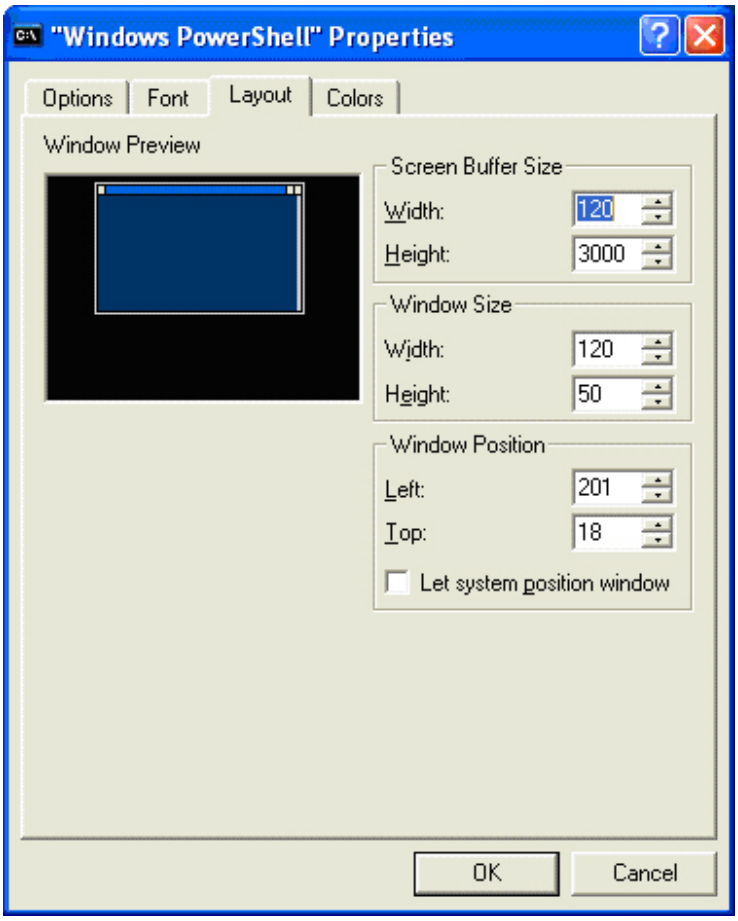
If you happen to be one of those people, we have good news for you: we're not sure any of this qualifies as "tricking out," but you *can* custom-tailor Windows PowerShell to a surprising degree. Don't like commands named Get-ExecutionPolicy and Set-AuthenticodeSignature? Fine; just **create a new alias**[1] and give these commands any name you want to give them. Don't like the default format PowerShell uses when displaying information for a particular type of object? Fine; **create your own .PS1XML file**[2] (as seen in the Windows PowerShell Week webcast) and change the default format. Don't like the way the Windows PowerShell console looks? Fine; change the console size, change the console fonts or colors, change pretty much whatever you want. It's entirely up to you. And, as it turns out, modifying the look of the console window is remarkably easy.

As we all know, Windows PowerShell is brand-new, cutting-edge technology; nevertheless, this new functionality is hosted within the same console window that William Shakespeare used when writing *Hamlet*. (That is, the same console window used by Cmd.exe.) The Scripting Guys are the first to admit that there are some definite disadvantages to this. For one thing, don't bother trying to use Ctrl+C and Ctrl+V to copy and paste text inside the console window; it's not going to work. However, there is at least one *advantage* to reusing the same shell as Cmd.exe: if you know how to modify the Cmd.exe console then you already know how to modify the Windows PowerShell console.

Point taken: what if you *don't* know how to modify the Cmd.exe console? That's OK; in just a second we'll show you how to use the GUI to modify the console window. And then, just for the heck of it, we'll show you some programmatic ways to modify console properties. You say you don't like the way your PowerShell window looks? Then read on.

Let's kick things off by showing you how to use the GUI to modify the console window. (As a bonus, you can use this approach to modify Cmd.exe as well as PowerShell.) To begin with, start Windows PowerShell. (Always a good place to start.) When the PowerShell window appears, click the icon in the upper left-hand

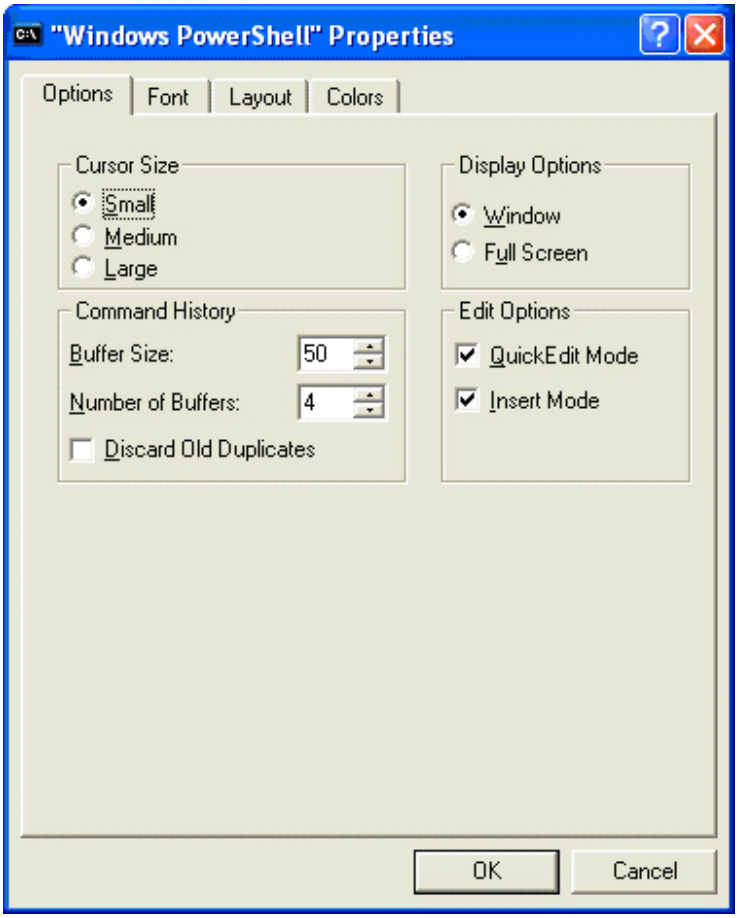
corner and the click **Properties**. That will bring up a dialog box that looks like this:



Recognize that? From here you can start clicking the various tabs - and start modifying the console window - to your heart's content. If you're not sure what some of these options are for then click the **?** icon in the upper right-hand corner of the screen and then click the option you're unsure about:

Note: If you're running Windows Vista you won't see the ? icon. You'll just have to guess at what everything does and hope you're not too surprised by the outcome.

We're not going to tell you how big or how colorful you should make your console window; that's up to you. However, we *will* recommend that you click on the **Options** tab and select both **Quick Ed it Mode** and **Insert Mode**:



Why? Well, QuickEdit Mode enables you to use the mouse to copy and paste within the PowerShell window. Assuming this mode is enabled, you can copy something to the Clipboard by highlighting the text with the mouse and then pressing ENTER; to paste text from the Clipboard, position the cursor and then click the right-mouse button. Insert Mode, meanwhile, enables you to insert text when typing. If Insert Mode is disabled then new text that you type in will overwrite any existing text. (Give it a try and you'll see what we mean.)

Tip. Here's something cool. Suppose you have a PowerShell script, either a simple one that looks like this or a more complicated example:

```
cd C:\Scripts
Get-ChildItem -recurse
```

Now, suppose you're looking at this script in Notepad or on the Web. Go ahead and highlight both lines, switch over to PowerShell, and then paste in the script. PowerShell will go ahead and run all the commands: it will change the working folder to C:\Scripts and then run the Get-ChildItem cmdlet. The point of all that? You can paste in an entire script and PowerShell will dutifully execute each line in turn; you do *not* have to paste a script in line-by-line.

When you're finished making all your changes click **OK**; that will bring up the following dialog box:

If you'd like to make these changes "permanent" (after all, you can always change your changes later), select

Modify shortcut that started this window. If you do that, the next time you start PowerShell (or at least the next time you start PowerShell using this same shortcut) you'll get your cool new custom look. If you select **Apply properties to current window only** your current console window will be modified, but all those changes will be lost when you end this particular PowerShell session.

Granted, it's not quite modding or skinning. But at least it's something, right?

Top of page

Using a Script to Modify Console Properties

Now, it's great that we can start PowerShell and then use a series of dialog boxes to modify the console window. However, we're script writers: we hate to do *anything* by hand if there's a way to do it using a script. So here's the question: can we modify properties of the console window programmatically?

As it turns out, there are several properties of the console window that are easy to modify using a script. (You can modify some of the not-so-easy to modify properties as well, but because that involves making changes to the registry we'll skip those for now.) In particular, you can easily change the console window colors (both the color of the window itself and the color of the text); the console window title; and the size of the window.

> **Note**. So why would you *want* to programmatically change the properties of the console window? Well, the truth is, maybe you don't. On the other hand, suppose you have multiple PowerShell sessions running at the same time. (Something you not only *can* do but sooner or later *will* do.) Changing the window title and/or changing the window colors makes it much easier for you to differentiate between Session A, Session B, and Session C. Likewise, modifying the window size helps ensure that your data will fit just the way you need it to.

The secret to programmatically modifying console properties is to use the **Get-Host** cmdlet. Typically, Get-Host is used to display information about PowerShell itself, particularly the version number and regional information. For example, type **Get-Host** from the PowerShell command prompt and you'll get back information similar to this:

```
Name             : ConsoleHost
Version          : 1.0.0.0
InstanceId       : 89c72fa7-c7f0-4766-8615-451990b15f70
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : en-US
CurrentUICulture : en-US
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
```

So what's the big deal here? The big deal is the **UI** property, which is actually a portal to a child object (which, as you can see from the output above, is derived from the .NET Framework class System.Management.Automation.Internal.Host.InternalHostUserInterface). The UI object, in turn, has a property named **RawUI**, which gives us access to console properties such as window colors and title. Take a look at what we get back when we run the command **(Get-Host).UI.RawUI**:

```
ForegroundColor       : DarkYellow
BackgroundColor       : DarkMagenta
CursorPosition        : 0,125
WindowPosition        : 0,76
CursorSize            : 25
BufferSize            : 120,3000
WindowSize            : 120,50
MaxWindowSize         : 120,82
```

```
MaxPhysicalWindowSize : 175,82
KeyAvailable          : False
WindowTitle           : Windows PowerShell
```

Many of these properties can be changed using a script. *How* can they be changed using a script? We were just about to show you.

But first, another question: Why the parentheses in the command **(Get-Host).UI.RawUI**? Well, any time Windows PowerShell parses a command, it performs operations in parentheses before it does anything else. In this case, that means that PowerShell is going to first run the Get-Host cmdlet and then, after that command has completed, access the UI property of the returned object and then access the RawUI property of the UI object (which happens to be yet another object). The single line of code **(Get-Host).UI.RawUI** is shorthand for the following:

```
$a = Get-Host
$b = $a.UI
$c = $b.RawUI
```

But who cares about the technical details, right? Let's do something fun, like change the background color and the foreground (text) color of the console window. Here's a three-line script that gives us yellow text on a green background:

```
$a = (Get-Host).UI.RawUI
$a.BackgroundColor = "green"
$a.ForegroundColor = "yellow"
```

Note. After running the script you might want to call the **Clear-Host** function (or its alias, **cls**) in order to "refresh" the screen. Otherwise only part of the window will feature yellow text on a red background.

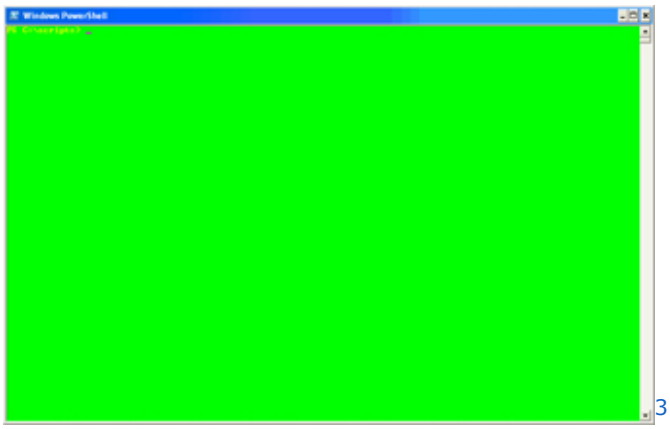
As you can see, there isn't much to the script. We create an object reference (named $a) to the UI.RawUI object, and then we simply assign new values to the **BackgroundColor** and **ForegroundColor** properties.

Speaking of which, here are the different colors available to you:

- Black
- DarkBlue
- DarkGreen
- DarkCyan
- DarkRed
- DarkMagenta
- DarkYellow
- Gray
- DarkGray
- Blue
- Green
- Cyan

- Red
- Magenta
- Yellow
- White

And here's what our new console looks like, in all its glory:



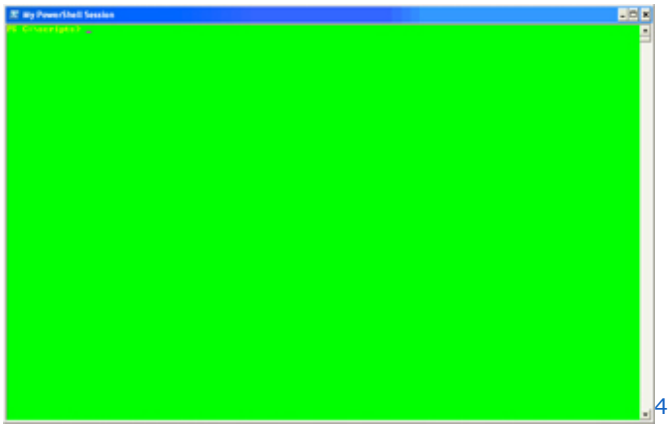
3

And yes, it *is* a good idea to wear eye protection if you choose this color scheme.

That was pretty cool, wasn't it? Now, let's see if we can change the window title:

```
$a = (Get-Host).UI.RawUI
$a.WindowTitle = "My PowerShell Session"
```

Wow; that took just *two* lines of code. Again we create an object reference to the UI.RawUI object. And then this time we assign a new value to the **WindowTitle** property. The net result? This:



4

Let's make one more little change before we call it a day: let's change the size of the console window itself. First let's take a look at the code, then we'll explain how it all works:

```
$a = (Get-Host).UI.RawUI
$b = $a.WindowSize
$b.Width = 40
$b.Height = 10
$a.WindowSize = $b
```

This is a tiny bit more complicated, simply because the **WindowSize** property is yet another object. In our first line of code we once again create a reference to the UI.RawUI object. We then grab the WindowSize

property of that object and assign it to a new object ($b). That's what this line of code is for:

```
$b = $a.WindowSize
```

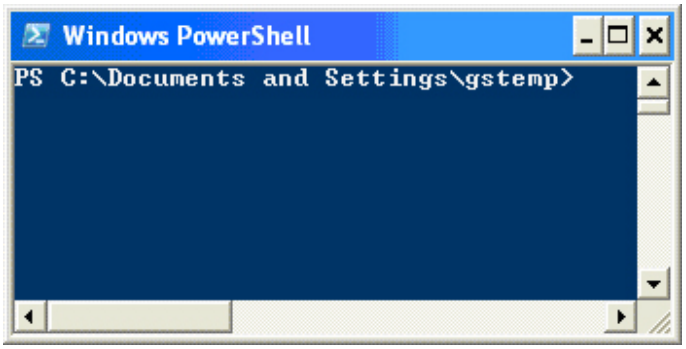
That gives us access to all the properties of WindowSize. At this point we can go ahead and assign new values to two of these properties, **Width** and **Height,** to $b. That's what we do here:

```
$b.Width = 40
$b.Height = 10
```

And then once we have $b's properties configured we can go ahead and assign $b (and its values) to the WindowSize property of $a:

```
$a.WindowSize = $b
```

Granted, it's not the most intuitive thing in the world. But it works:



Top of page

Modifying Your Windows PowerShell Profile

So let's assume you create the perfect console window: it's got the right colors and the right title, and the window is sized perfectly. How can you make sure that you get these same settings each and every time you run Windows PowerShell?

Here's one way: put the appropriate commands into your PowerShell profile. A profile is like an autoexec.bat file: it's a script file that gets executed each time you start PowerShell, (OK, technically that's not true: it's possible to start PowerShell without loading a profile, and it's possible to have multiple PowerShell profiles. But we're going to ignore those possibilities for now.) Put your console-modification code into your profile and you'll get the same, cool console window each time you start PowerShell.

And don't worry; we were just about to tell you how to customize your profile. To call up your default user profile start PowerShell and then type the following at the command prompt:

```
notepad $profile
```

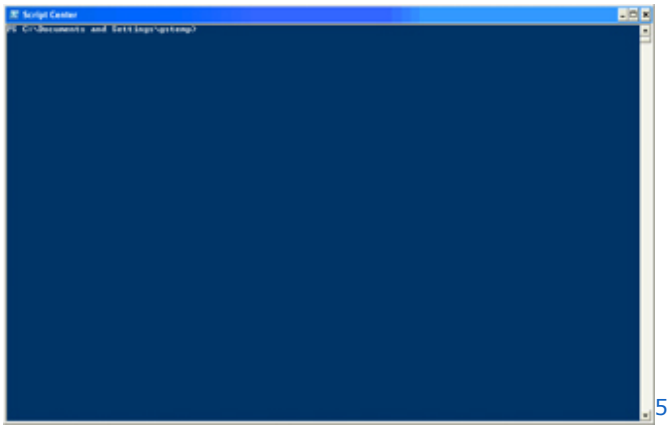
This will start Notepad with your default profile preloaded.

> **Note:** Don't panic if Notepad starts up and your default profile in blank: your profile *will* be blank, at least until you add something to it. Also don't panic if you're running Windows Vista and your profile doesn't open at all. We'll explain that in a minute.

At this point all you have to do is enter the desired commands and then save the file. For example, suppose you want your console window to always have the title **Script Center** (which sounds like something most people *would* want). Okey-doke; bring up your profile, type in the following commands, and then save the file:

```
$a = (Get-Host).UI.RawUI
$a.WindowTitle = "Script Center"
```

Now restart PowerShell and take a gander at the window title:



5

Pretty sweet, huh?

Working With the Default Profile in Windows Vista

Unfortunately, things get a little more difficult in Windows Vista. If you type **notepad $profile** before you've created a profile in Windows Vista, you'll receive a message box that says "The system cannot find the path specified." Your only choice is to click OK, at which point you'll have a blank instance of Notepad open but no profile file. To find out where your profile *should* be, simply type this at the command prompt:

```
$profile
```

This will return something like this:

```
C:\Users\kenmyer\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
```

Simply save the Notepad file to this location (you might have to create the WindowsPowerShell folder), with this name (Microsoft.PowerShell_profile.ps1) and you'll have your profile. At this point you can continue on with the rest of these instructions.

Top of page

And There's More to Come … Well, One of These Days

As we noted at the beginning of this article, there's even *more* you can do to customize PowerShell: you can create your own aliases; you can change the way certain data is displayed; you can add your own data types; change the PowerShell prompt; and so on. We won't make any promises, but we'll try to get to all these topics, and more, in the future. In the meantime, you might to take a look at the Script Center article on **tab expansion**[6]. Also, check out the **Windows PowerShell team blog**[7]. This blog is targeted towards the more experienced PowerShell users, but even as a beginner you should find plenty of useful and interesting information there.

Top of page

Windows PowerShell Owner's Manual

- Getting Started with Windows PowerShell[8]
- Windows PowerShell Shortcut Keys[9]
- Piping and the Pipeline[10]
- Running Windows PowerShell Scripts[11]
- Windows PowerShell Profile[12]
- Windows PowerShell Aliases[13]

Top of page

Links Table

[1]*http://technet.microsoft.com/en-us/library/ee176913.aspx*

[2]*https://www.livemeeting.com/cc/mseventsbmo/view?id=1032313506&role=attend&pw=48928A1E*

[3]*http://technet.microsoft.com/en-us/library/Ee156814.console5(l=en-us).jpg*

[4]*http://technet.microsoft.com/en-us/library/Ee156814.console6(l=en-us).jpg*

[5]*http://technet.microsoft.com/en-us/library/Ee156814.console8(l=en-us).jpg*

[6]*http://technet.microsoft.com/en-us/library/dd315316.aspx*

[7]*http://blogs.msdn.com/powershell/default.aspx*

[8]*http://technet.microsoft.com/en-us/library/ee177003.aspx*

[9]*http://technet.microsoft.com/en-us/library/ee176868.aspx*

[10]*http://technet.microsoft.com/en-us/library/ee176927.aspx*

[11]*http://technet.microsoft.com/en-us/library/ee176949.aspx*

[12]*http://technet.microsoft.com/en-us/library/ee692764.aspx*

[13]*http://technet.microsoft.com/en-us/library/ee692685.aspx*

Community Content