



Quick Guide

Adding a graphical interface to your PowerShell scripts is easy, fun and free with PrimalForms Community Edition/

Creating Graphical PowerShell Tools with PrimalForms Community Edition

In the past, creating a PowerShell script using Windows Forms to create a graphical interface was a very tedious task and rarely used except for the most simplest of forms. No more. The *free* PrimalForms Community Edition tool lets you create very rich Windows Forms-based PowerShell scripts using a WYSIWYG editor. Drag and drop the form elements, define your event handlers, export the form to a file,

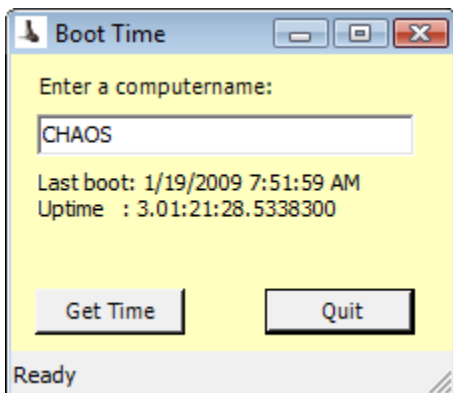
next to the computer and displays when the computer last booted and how long it has been running. Here's how easy it is to create a script like this.

Before we can begin you'll need to go to the Downloads section of www.PrimalTools.com and get PrimalForms. I'm assuming you already have PowerShell installed. PrimalForms will work with either PowerShell v1.0 or v2.0.

the property panel for each control that you drag and drop.

Click the form itself. In the property panel you will likely want to change the default form title, *Primal Form* to something else. Edit the Text property. I changed mine to *Boot Time*. You can also change the BackColor property as I did to yellow.

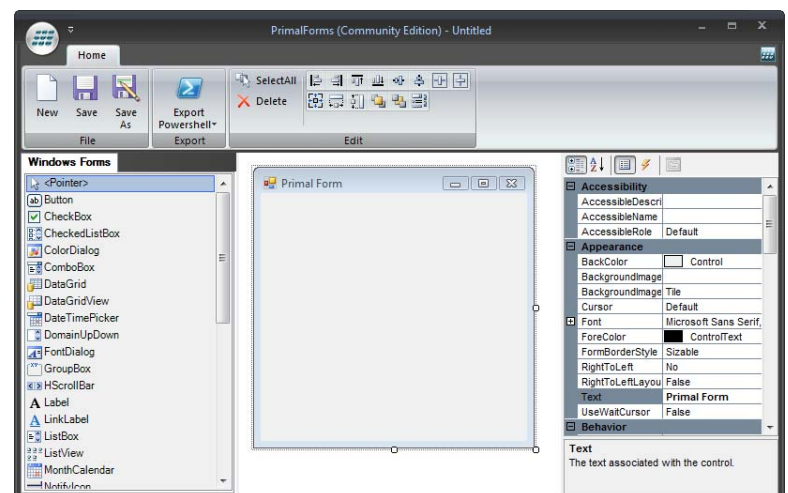
Drag a label control to the form. Notice you can drag and reposition anywhere on the form. Change the text to: *Enter a computername:*. Of course we need



add your PowerShell code and run your script. The Boot Time form you see is a PowerShell script I created with PrimalForms. The script uses the computername in the text box to con-

Creating the Form

Start PrimalForms and click New to create a new form. On the left side are all of the form elements such as buttons and text boxes that you can add to your form. On the right side is

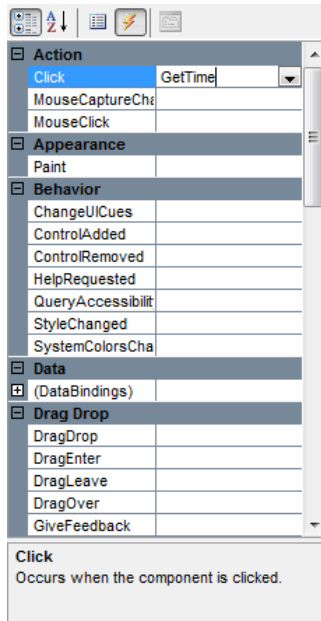




some place for the user to enter a computer name. For that we'll use a TextBox. Grab one and drag it onto the form. Position and resize as you wish.

If you look at the property panel for the text box, you'll see it has a default name like *TextBox1*. I prefer to give controls more meaningful names because I will reference this object in the PowerShell script by this name. Something like *txtComputername* is more meaningful because I can tell from looking at the name that it is a text box control and I have a better idea of which one. When the PowerShell code is generated for the form, this property name will become a variable named *\$txtComputername*.

I'm also going to give the text box a default value of the local computername. Because I'll be running a PowerShell script, I can use PowerShell expressions like `$env:computername` which will return the value for the `%Computername%` environmental variable.



Next I need some buttons for the user to click, so drag a button control and position it on the form. As with the label control, I change the control name from *button1* to something like *btnGo*. Change the text that appears on the button as well. I created buttons to get boot time information and to close the form. But what happens when a user clicks the button? When that happens, a Click event is fired. It's up to you to decide what happens.

Select the Go button. On the property panel click the lightning bolt icon. This will display the event handler properties. You may see different properties depending on the selected control. For a button, enter something for the Click event. This will become the name of the corresponding PowerShell scriptblock. In the graphic above I've defined a scriptblock called *GetTime*. Later I'll add PowerShell commands in my script editor to this scriptblock.

Technically you don't have to define a click event handler, PrimalForms will automatically create one for each button, but I prefer to use more descriptive names instead of *btnGo_OnClick*. But you don't have to.

You can define events for all sorts of actions such as selecting an item from a drop down box to when a form is first shown.

The last item to add to the form is a place to display the results. For that I'll use a Rich Text Box control positioned underneath the text box. If you don't want the border, simply change the *BorderStyle* property to *None*.

It doesn't take long to lay out the form controls, define properties and events. It's now time to add PowerShell code to make the form do something. PrimalForms Community Edition will generate a PowerShell script to display the graphical Windows Form elements which would otherwise be an extremely tedious process if you had to manually develop the script from scratch.

You can either export the form to a PowerShell script file, export it to the clipboard, or if you have PrimalScript 2009 installed, export to a file and edit.

Adding PowerShell

When you look at the PrimalForm generated script, most of the code is for generating the form elements. You shouldn't have to edit any of that code. Instead, you need to create the code for your event handlers, which I'll cover in a moment.



The script is essentially a function that runs the form. You can insert any other functions or PowerShell commands you require when you begin developing more advanced scripts. But for now all we need to edit is a scriptblock like this:

```
#-----
#Generated Event Script Blocks
#-----
#Provide Custom Code for events specified in PrimalForms.
$GetBootTime=
{
#TODO: Place custom script here
}
```

You can write whatever PowerShell code you want within the scriptblock. You can access the form controls and their properties as well. In the script you'll find variables defined for all of the controls.

```
#region Generated Form Objects
$form1 = New-Object System.Windows.Forms.Form
$statusBar1 = New-Object System.Windows.Forms.StatusBar
$rtbResults = New-Object System.Windows.Forms.RichTextBox
$btnGo = New-Object System.Windows.Forms.Button
$txtComputername = New-Object System.Windows.Forms.TextBox
$label1 = New-Object System.Windows.Forms.Label
$btnQuit = New-Object System.Windows.Forms.Button
$InitialFormWindowState = New-Object System.Windows.Forms.FormWindowState
#endregion Generated Form Objects
```

PrimalScript will provide PrimalSense for any defined WinForm element which makes script development pretty easy. PrimalForms Community Edition includes a Getting Started Guide with a number of sample forms and scripts which you can use as starting points for your own projects or as teaching examples.

If you realize you need to modify the graphical elements you'll need to reopen your PrimalForms project, generate a new script and re-add your PowerShell code.

However there are a few techniques for creating scripts and moving between the code gener-

ated by PrimalForms Community Edition and the script you create in PrimalScript. One technique that I've started using is to leverage PrimalScript's snippet feature.

Like many of you I'll start with a basic form and begin building a script. Inevitably I have to go

back and modify the form. Before I do so, I select my script block section, right click, and use the Save as Snippet feature. In PrimalForms, I copy the script to the clipboard, return to PrimalScript and paste into my script file. I select and delete the script block. In its place I insert my saved snippet. A variation is to use PrimalScript's clipboard

ring, by first copying my script block, then using Ctrl+Shift+V to cycle through until I find the text I need.

Another technique comes from the SAPIEN support forums. Create a script file from PrimalForms. The next time you need to modify the form and script, open the script in PrimalScript 2009. Modify the form and export the result to a file. In PrimalScript, use the compare files feature to compare your script with the exported file. PrimalScript will highlight all the differences. All you have to do is copy and paste the differences from the exported file to your script. This can be made even easier by splitting the window so that you can see your script and the compare files tab. Select text then drag and drop it into your script.

When creating your scriptblocks, I recommend first developing a standalone PowerShell script or function. Once you know your PowerShell code works, you can insert it into your PrimalForms' script and edit as necessary to work with the graphical controls. This is important because if the forms based version of the script has problems I can trust that my basic PowerShell expressions are valid and problems are most likely related to using the form elements.



Quick Guide

For the boot time form I already had a short PowerShell script that worked. Here's my original code:

```

$computername=$env:computername

$os=Get-WmiObject win32_operatingsystem -computername $computername

Write-Host ("Last boot: {0}" -f $os.ConvertToDateTime($os.lastbootuptime))
Write-Host ("Uptime : {0}" -f ((get-date) - os.ConvertToDateTime($os.lastbootuptime)).tostring())

```

quired wherever the file will be running.

When you are ready, not only can you create a standalone PowerShell script, but you can also package it into an executable di-

rectly from PrimalForms 2009.

I will modify this code and add it to the script block that executes when the Get Time button is clicked. Modifications are pretty simple. I need to use the value from the text box for the \$computername variable.

```

$computername=$txtComputername.Text
$os=Get-WmiObject win32_operatingsystem -computername $computername -ea SilentlyContinue

```

PrimalForms 2009

PrimalForms Community Edition is terrific at generating PowerShell scripts that display Windows forms and controls. However, adding PowerShell code for the associated events requires a script editor like PrimalScript.

If you'd like a more integrated solution take a look at PrimalForms 2009, the commercial cousin to the Community Edition.

The same **Get-WMIObject** expression is executed. However instead of writing results to the console, the results will be written to the rich text box control.

```

$line1="Last boot: {0}" -f
$os.ConvertToDateTime($os.lastbootuptime)
$line2=$rtbResults.text="Uptime : {0}" -f ((get-date)
- $os.ConvertToDateTime($os.lastbootuptime)).tostring()
$rtbResults.Text=$line1+ "`n" + $line2

```

I save the script and test it in PowerShell. Now instead of a plain console based script I have a graphical tool I can deploy to any machine configured to run PowerShell scripts. I can even use PrimalScript's Script Packager to create an exe file, although PowerShell is still re-

quired wherever the file will be running. When you are ready, not only can you create a standalone PowerShell script, but you can also package it into an executable directly from PrimalForms 2009. If you'd like to see more, download an evaluation copy from PrimalTools.com. As always, if you require some assistance developing your PowerShell Script please join us at ScriptingAnswers.com.

If you'd like to see more, download an evaluation copy from PrimalTools.com. As always, if you require some assistance developing your PowerShell Script please join us at ScriptingAnswers.com.

Jeffery Hicks

