**How to Create or Edit a Module in DSCntrl**

**Requirements:**

* **Windows Computer**
* **Visual Studio 2019**
* **Code/Text Editor (I use Visual Studio Code)**

**Running DSCntrl**

In order to run the program, navigate into the DSCntrl folder, and then run the shortcut named DSCTRL or run the ds.ps1 file. On some systems it may be easier to use the shortcut as running the file directly may be blocked by the execution policy of the system. If you are having issues running the .ps1 file directly, try opening up a PowerShell window and running the command:

Set-ExecutionPolicy Unrestricted

This may require administrative permissions, and if the computer is joined to a domain, the Group Policies may not allow this setting to be changed. Also keep in mind that this will reduce your computers security to PowerShell script files, so be extra cautious when dealing with unknown ps1 files. If you would like to set your execution policy back to the default you can run the command:

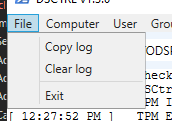
Set-ExecutionPolicy RemoteSigned

**Adding a Menu Item to the Main Interface**

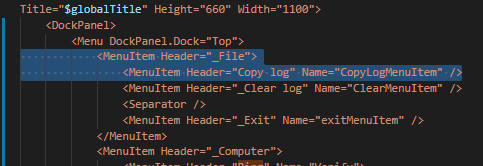
In order to make changes to the main DSCtrl interface, open the *DSForm.ps1* file. At the top there will be a very large set of XML code – this is what you will be editing. There is a structure of menus and menu items, the upper level menu items are the menu buttons at the top of the screen.



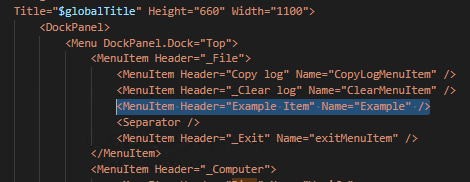
Within that are sub-menu items which are the options within each menu.



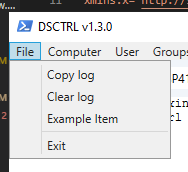
So in this case, the “File” menu item has the sub menu item “Copy log”, which is located here within the XML:



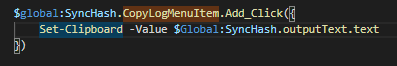
If you would like to add a new menu item to the interface, you would need to find under which menu you would like to add it to, in this case, if we wanted to add a new menu item to the “File” menu, you would add a new line anywhere within that menu item. Then, for simplicities sake, copy the line above it, paste it into the new line, and then edit the information accordingly. Like this:



Which appears in the interface like this:



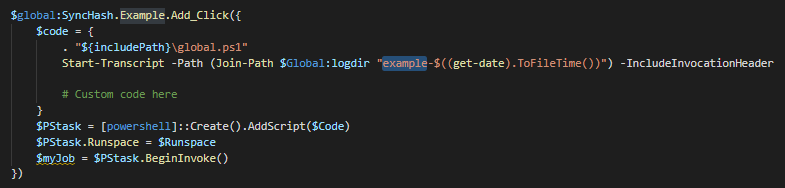
After creating the menu item, you will need to then write the code that will be run when the menu item is clicked. Scrolling down within DSForm.ps1, there will start to be blocks of code that handle on click events, the block that handles the “Copy log” from the example above, looks like this:



Any on click event handler added will be in this format - $global:SyncHash.<name of menu item>.Add\_Click({<Code to execute goes here>}), so for the example menu item that we added, since we gave it the name “Example” we would then handle the on click function with a block that looks like this:

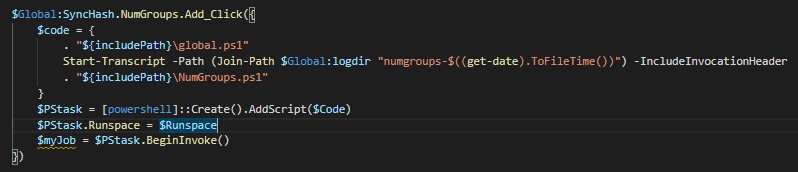


In the example above, when the user clicks the menu only one line of simple code is executed, but this can be as complicated as needed to handle whatever task you would like it to handle. Although the more complex code tends to follow similar formats within the file, which looks like this:



This ensures that the global file (which holds a lot of global information, think of it like importing a module in another coding language) is run, and a transcript is started. Note that the transcript name needs to be changed – in this case it’s “example-$((get-date).ToFileTime())” instead of say “copylog-$((get-date).ToFileTime())”. After the transcript has been started, you can add any powershell code afterwards as normal, then the code will be executed as a task within its own run space.

One way to make this project more modular is to not write code directly within this block, but instead write a separate .ps1 script file, and then call the script file from here like this example:

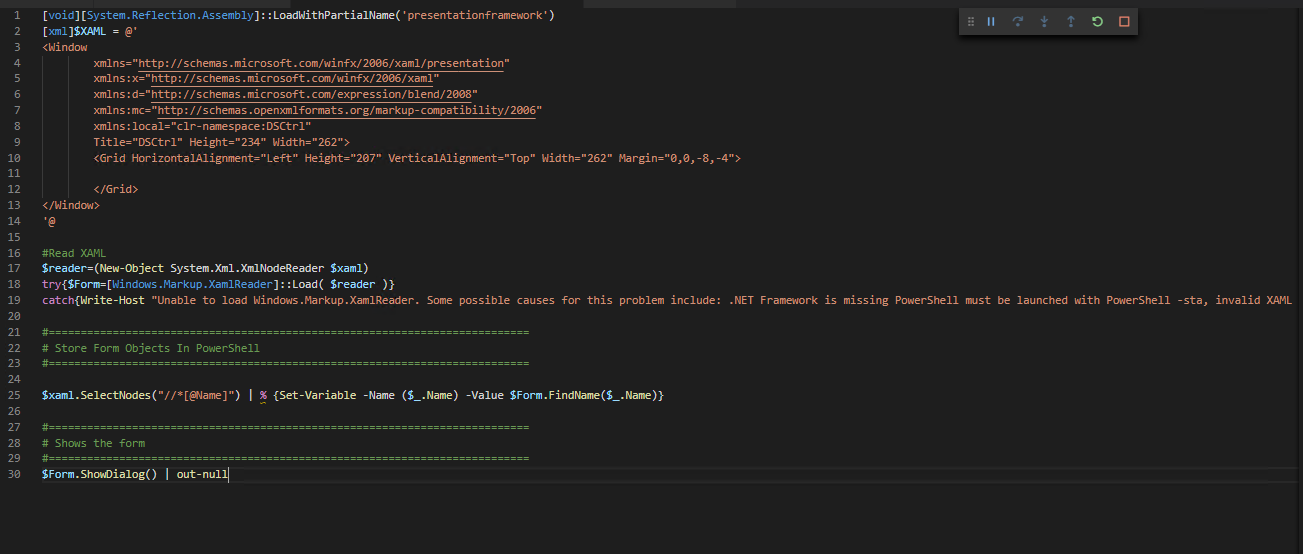


This can help keep from clogging up the main form file and keep scripts more modular. One goal of this capstone is to clean up the DSForm.ps1 file to be as concise as possible by converting as much code within the file to their own separate script files.

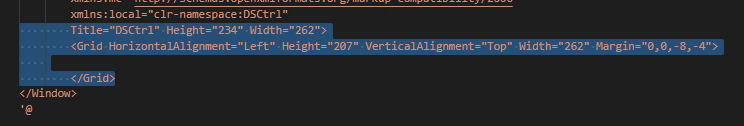
**Creating an XML Window**

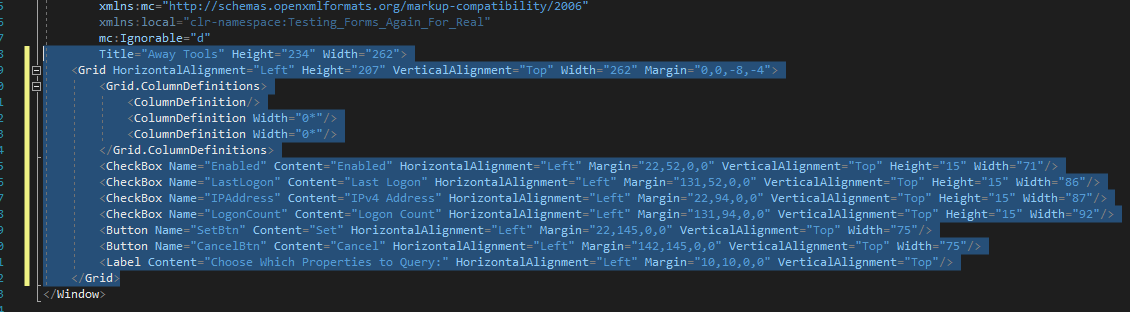
By using Visual Studio 2019, it’s possible to create your own XML windows that can be used to gather and present information to the user, however, there are a few key pieces that you will need to ensure are adhered to in order to make it as easy as possible on you. The first important note is that, when you create a project within Visual Studio 2019, there will be some syntax within the XML that will not play well with PowerShell, so to make it easier on myself I tend to copy and paste either the XML template or copy and paste the XML I am working with back and forth between Visual Studio 2019 and Visual Studio Code. This can be confusing, but here is an example.

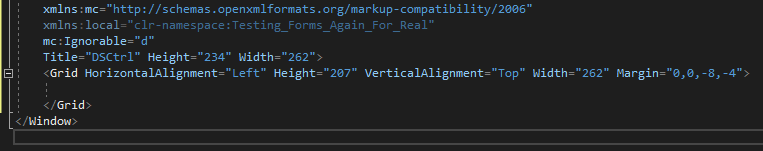
First, I will start with the XML Template that I have created (be sure not to save over this file, just copy and paste it into your new ps1 module file).



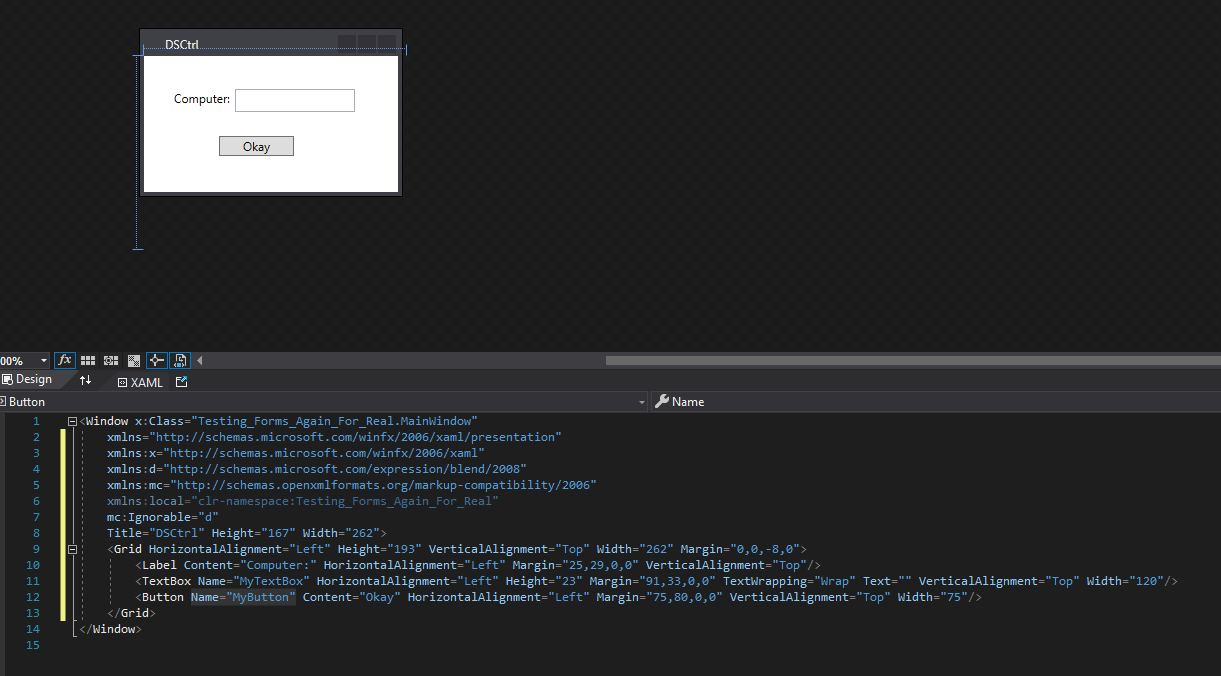
This template contains all of the important base XML and powershell to run a separate window. I will then copy over everything from the line within the XML starting with Title=” down to the final grid element, like this:



I will then go into Visual Studio 2019, where I will overwrite the information within the same elements. 

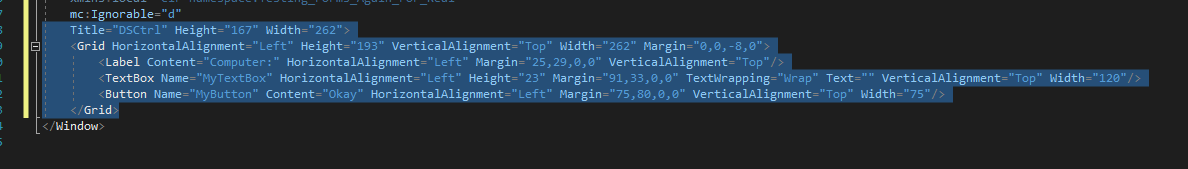


From here I will use the drag and drop features within Visual Studio 2019 to create the interface that my module requires.

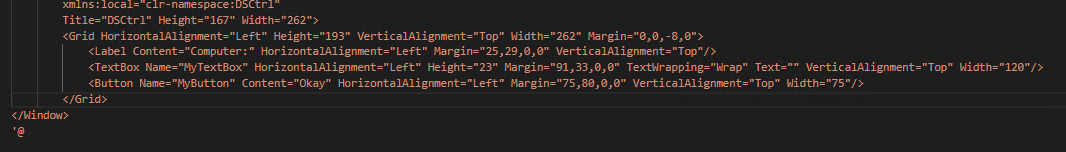


Make sure that you add the Name=”<yournamehere” attribute to any item you add to the window that you want to pull information from or interact with. In this case, I have a textbox named “MyTextBox” and a button named “MyButton” that I want to interact with.

Then I will use the same copy method from before, but overwrite the information within the .ps1 module with the updated XML. For example:



Over to Visual Studio Code (where I am working on the modules .ps1 file)

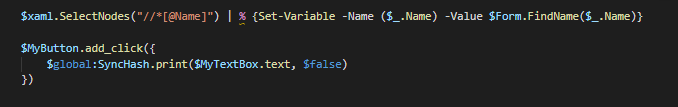


After that, it’s time to code some functionality to the window using the button and the text box, but first we need to take a look at one very important line of code that we copied from the template file:



This line of code needs to precede any interaction can take place via powershell, since this is what parses the XML code and assigns powershell variables to the named elements within the XML. So since we have a “MyTextBox” and a “MyButton” within the XML, we now have the $MyTextBox and the $MyButton variables that refer to that text box and that button specifically.

Now you can interact with the items within the window, for example:



We can add a click event for the $MyButton button, so when it’s clicked it will print the text from the $MyTextBox field. In this example we are using the custom print function that allows us to print information back to the main window.

To tie this back into the “Creating a Menu Item” portion, we can now call this script from the example menu item we created.



So when we run the program, and select the Example Item menu within File, we will be presented with the window we created, which we can then interact with.

