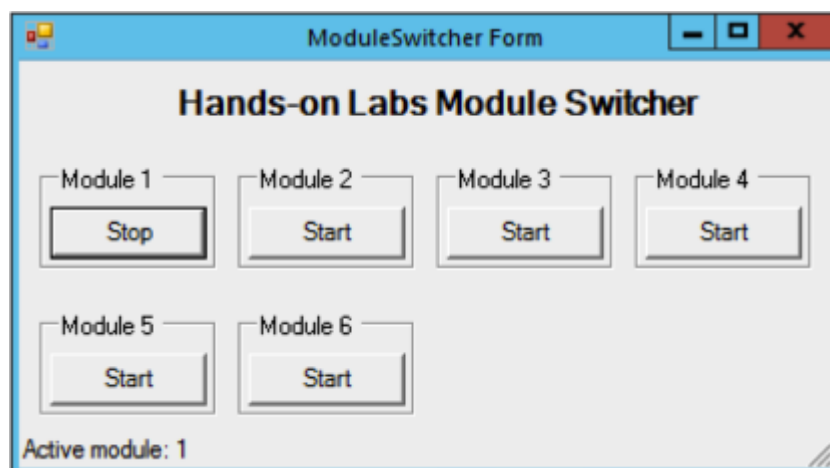


HOL Module Switcher

This document is intended to outline the intended functionality of the *Hands-on Labs Module Switcher* script, explain its current functionality and provide documentation of its configuration, implementation, and use.

Background



VMware Hands-on Labs are made up of a lab environment (a “vPod”) paired with a manual. Each vPod may be paired with multiple manuals, and each manual may contain multiple *modules*. When a vPods starts up, it can only enter one *Ready* state, and it would be up to the user to get the vPod from the base Ready state to any other state expected by the manual or module that is being used.

For example, consider that a user wants to jump into a lab at Module #3. During the course of the lab work in Modules 1-2, a user may have been tasked with making changes to the environment as part of the lab exercises. Once Module 3 is reached, there may be an expectation that certain virtual machines have been powered up, powered down, or otherwise reconfigured from the base state. While not a strict dependency, the workflow of the lab may leave the lab environment in a different state. At the very least, users may see differences between the console and screen shots provided in the manual.

There are generally two options to address such issues: have the user perform the necessary reconfiguration manually, or provide some form of automation to facilitate the process. In the VMware Hands-on Labs, we try to minimize non-essential inputs as much as possible to provide a better user experience, so going the automation route is preferred.

Goals and History

Standalone DOS/Windows batch files and PowerShell scripts have been used to provide “fast-forward” or reconfiguration capability within some of our labs. In other labs, a binary *HOL Optimizer* was used to provide a button dashboard of sorts. it was mostly limited to powering on and off vVMs and reverting snapshots of layer 2 VMs (vVMs), and it was not as open and extensible as we would

have liked. The goal of the *HOL Module Switcher* solution is to combine the best parts of these options while providing an open and manageable framework. I think this is summed up nicely by Bill Call: “*Simple. Generic. Better.*”

The guiding principle for the HOL Module Switcher is that it is to be kept as simple as possible so that it can be maintained with a minimum effort. A pure PowerShell solution was preferred over DOS/Windows batch or compiled binary for several reasons: access to the Windows graphical toolbox, ease of management, no need to install a compiler, and minimal overhead.

In addition, because this solution is written entirely in PowerShell, most of the functions the Hands-on Labs Core Team have created and tested to support the LabStartup script may be used to provide functionality to the the Module Switcher’s module scripts. These functions are contained within the *C:\HOL\LabStartupFunctions.ps1* file.

Starting with the VMworld 2016 development cycle, we have introduced the concept of co-op vPods. The Module Switcher can be used to move a shared vPod from its base running state into the start state required for whichever manual the user has selected. The Module Switcher has no knowledge of which manual a user has selected, but it is possible to have the user provide that information by selecting the Module Switcher configuration that matches the selected manual. More on this later.

Specifications

A vPod boots into a known base state. For the sake of simplicity, we will call this state *Module 0*. Every other module should be able to be reached from Module 0 state by applying specific scripted actions to the environment. This provides the capability for a user to *jump in* at any module within the lab: click the Module X Start button and you’re good to go with Module X once the scripts have been run.

As users progress through a lab manual, they should be instructed to activate the *START* action for the module that they would like to take prior to beginning other tasks for that module. Once the START action has completed preparation of the lab environment for the specified module, the user can proceed with the lab. As an example, many modules contain a few pages of expository material at the beginning of the module. Prior to presenting this material, the user would be instructed to activate the associated START action to have the script begin reconfiguration. While the user reads the material in the manual, the script can be performing its work. This minimizes user wait time and provides a good lab experience. Of course, if a user chooses to watch the script execute, there should be adequate feedback provided to them in the console as well ([see R. Grider, K. Luck WebEx presentation section regarding script output within labs](#)).

To keep implementation of the START/STOP actions as simple as possible, the Module Switcher limits its presentation of the ability to “roll back” the environment. When a user jumps into a lab module, all previous modules’ activation buttons will be disabled. Teams may elect to implement the ability to take any module in any order, however, this increases the number of test cases to manage and endangers lab sanity.

NOTE: Before you consider allowing “roll back,” please think about not only the impact on vVMs, but also the configuration changes to Layer 1 VMs that occur during the course of your lab. Some changes are difficult or impossible to roll back without causing significant

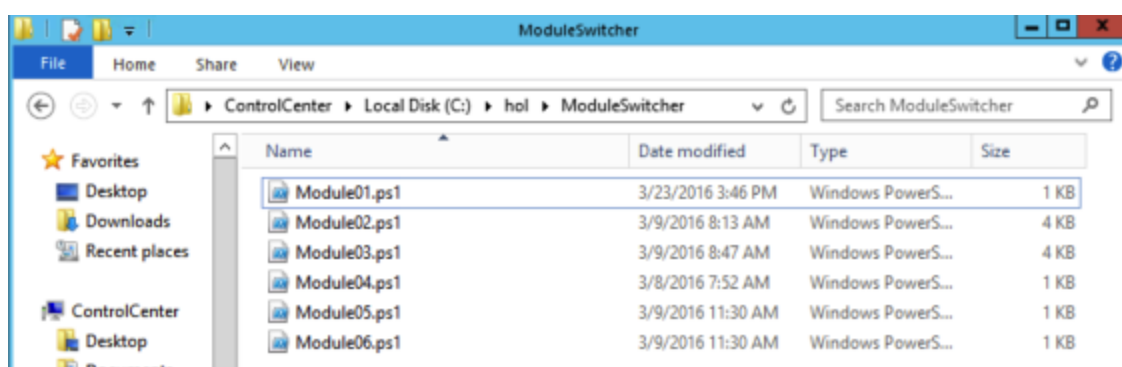
trauma to the lab environment. We are not able to roll back snapshots to Layer 1 VMs from within the lab.

Specifications Summary

For those who just want the highlights:

- We're using automation to provide "fast-forward" capability
- It is only necessary to support jumping forward, not backward
- PowerShell is the preferred automation, although anything that can be *called from* PowerShell will work
- The UI is intended to be simple, flexible and not require teams to modify its code unless they have specific needs
- Most of the prewritten functions in the LabStartupFunctions library are available for use in the module START/STOP code. This includes querying, starting and stopping services on Windows and Linux machines, copying files via SCP, executing commands via SSH, testing URLs, responses to pings, starting vVMs and vApps and more.

Module Switcher Details



To support this effort, the Hands-on Labs Core Team is providing the Module Switcher and the functions within LabStartupFunctions.ps1. The Module Switcher utility's core functions are to present a panel of buttons, one for each module, and facilitate the execution of lab team-provided START/STOP actions.

The Module Switcher is made up of the *ModuleSwitcher.ps1* file and a directory containing the scripts used to manage the environment. The default directory is *C:\HOL\ModuleSwitcher*, but that can be changed on the command line by providing the *-ModuleSwitchDirPath* switch and providing an alternate path. The ModuleSwitcher.ps1 file contains the form display, management, and script launching functionality. Typically, it will not be necessary to modify this script.

The *ModuleSwitchDirPath* (by default, *C:\HOL\ModuleSwitcher*) directory must contain one script file for each module in your lab, even if the script does nothing more than report that it ran. Having one file per module allows the Module Switcher to automatically configure the number of buttons in the window to match the number of modules in the lab.

The naming convention used is important: these are the names the ModuleSwitcher.ps1 script uses to enumerate and launch the scripts contained in the *ModuleSwitchDirPath*. The ModuleSwitcher automatically creates a Module group and a Start button for each Module##.ps1 script it finds.

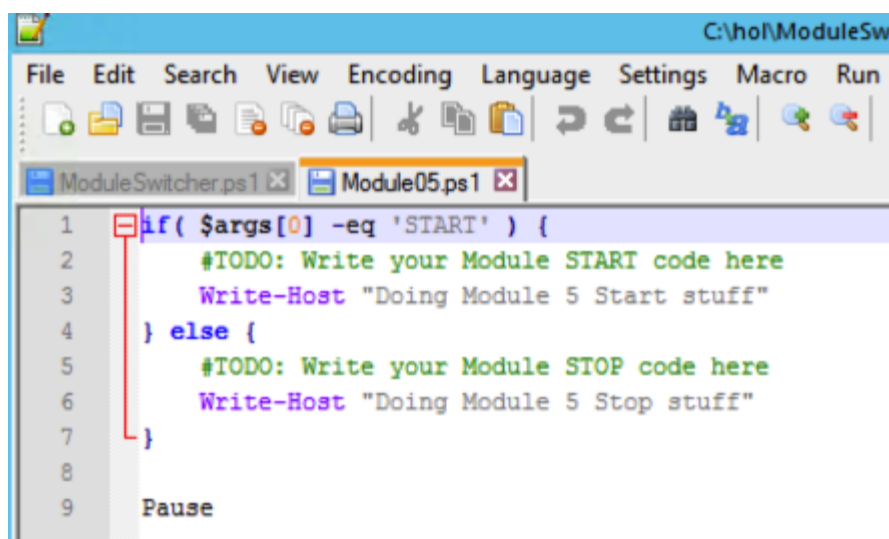
Note that two digits are used for the script numbering (“Module01.ps1” rather than “Module1.ps1”). This ensures that the scripts sort in the proper order when displayed in Explorer or are enumerated and sorted in PowerShell.

BYO Scripts

That is most of what you need to know about the Module Switcher. It may be helpful to look at it as a simple GUI menu system. The core functionality, the *switching*, is provided by your team in the form of the Module scripts’ START and STOP actions.

Ideally, anything that you need to do to check or alter the lab environment state such as URLs, services, etc. in preparation for a given module can be called using this mechanism. If it is something that cannot be done directly with PowerShell, there are other options, such as calling out to a Linux machine to execute a command line option, or calling a Windows/DOS batch file from a PowerShell context. Please let the core team know if you need assistance, and I am certain the SDK/API team would be interested in hearing about anything that you might be trying to do but are unable to do.

Module##.ps1 Script Contents



```
1  if( $args[0] -eq 'START' ) {
2      #TODO: Write your Module START code here
3      Write-Host "Doing Module 5 Start stuff"
4  } else {
5      #TODO: Write your Module STOP code here
6      Write-Host "Doing Module 5 Stop stuff"
7  }
8
9  Pause
```

Each *ModuleXX.ps1* script file in *C:\HOL\ModuleSwitcher* has a START and a default action. The simplest script looks something like the image in this step.

When ModuleSwitcher.ps1 calls the Module##.ps1 script for module ##, it will specify an action of START if the module is being started (the clicked button's text is "Start"). Any other call will trigger the STOP (default) action. At a high level, the START block should prepare the environment for the user to begin the specified module and the STOP block should undo those actions.

Please note the *PAUSE* at the end of the script. ModuleSwitcher.ps1 will call this script in its own PowerShell window so that its progress and actions can be reported to the user. The PAUSE will

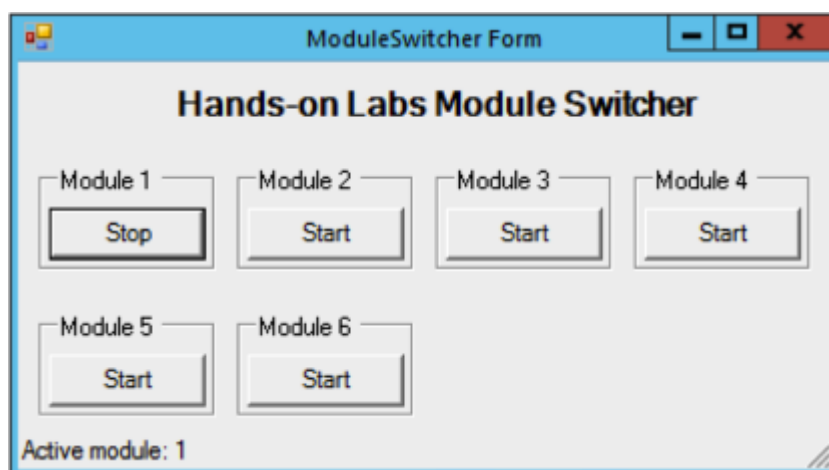
leave that information on the console until the user presses ENTER to continue. This is helpful in showing the user that the environment is ready for them to proceed, still working on it, or blocked for some reason.

Execution

Calling the Module Switcher is as simple as running the PowerShell script. It is a little cleaner for users if you wrap it in a batch file that calls PowerShell and provides the ModuleSwitcher.ps1 script as an argument. The following 2-line batch file will run the ModuleSwitcher.ps1 in a hidden PowerShell window so that only the main panel is visible. This has been provided as *C:\HOL\ModuleSwitcher.bat*

```
@echo off
C:\WINDOWS\system32\windowsPowerShell\v1.0\PowerShell.exe -windowstyle hidden "& 'C:\HOL\
ModuleSwitcher.ps1'"
```

Initial Run



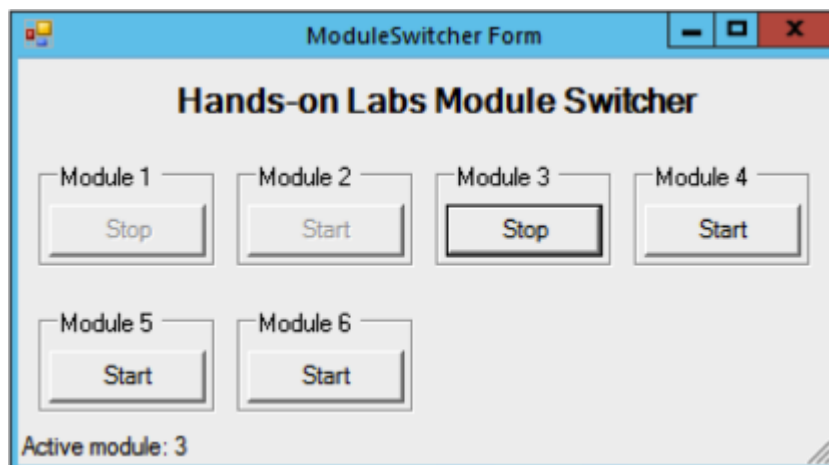
During the initial running of the ModuleSwitcher.ps1 script, it will look for files matching the "Module##.ps1" pattern in the *C:\HOL\ModuleSwitcher* directory. Based on the number of scripts found, a window will be created containing the required number of buttons, in numerical order, one for each module. Continuing our example, there are 6 scripts the directory.

NOTE: The default layout is 4 columns across, which should work well for most use cases. As of the time of this writing, the maximum supported layout is 4 columns by 5 rows, which will support 20 modules. If you require support for more than that, please let the core team know.

Each module is presented with a single Start button. Clicking the Start button for a module will first call the STOP action from the *currently active module*. Once that script has run and the user has pressed the ENTER key, the Module##.ps1 script for the selected module is called with the START action.

NOTE: As of version 1.19 of ModuleSwitcher, pods are assumed to enter the *Module 0* state at boot time. Prior versions assumed Module 1, but this behavior did not necessarily make sense for shared pods.

Jump to Module 3

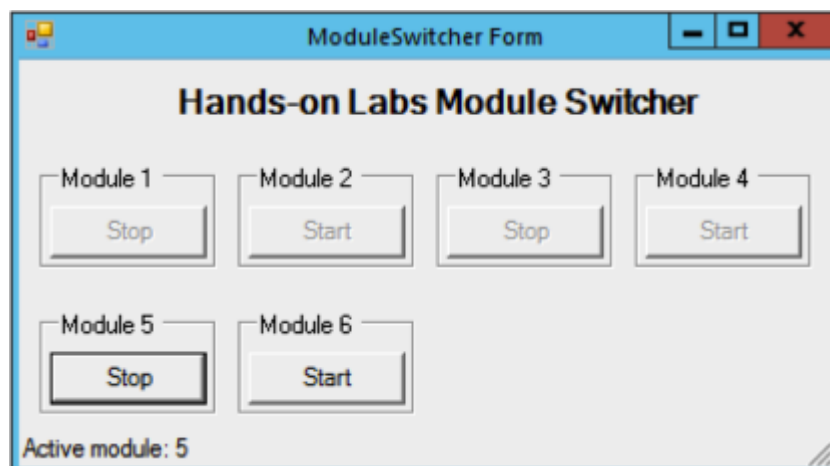


In this example, if a user wants to jump to Module 3 and clicks on the Start button in the Module 3 section, the Module01.ps1 script will be called with the STOP action, followed by the Module03.ps1 script with the START action. Why? Because Module 1 is listed as the current *active module* (see the status bar at the bottom of the window in the previous step's image), the script determines which module needs to be cleaned up prior to beginning the selected one.

Once the Module 3 START action has been processed, and the user presses the ENTER key to continue, the ModuleSwitcher buttons look like the image in this step.

Note that the Module 1 and Module 2 buttons are now disabled (greyed out) because they have been either completed or bypassed. The Module 3 button changes from *Start* to *Stop* and the Active module has been updated to show module 3 in the status bar at the bottom of the window.

Jump to Module 5



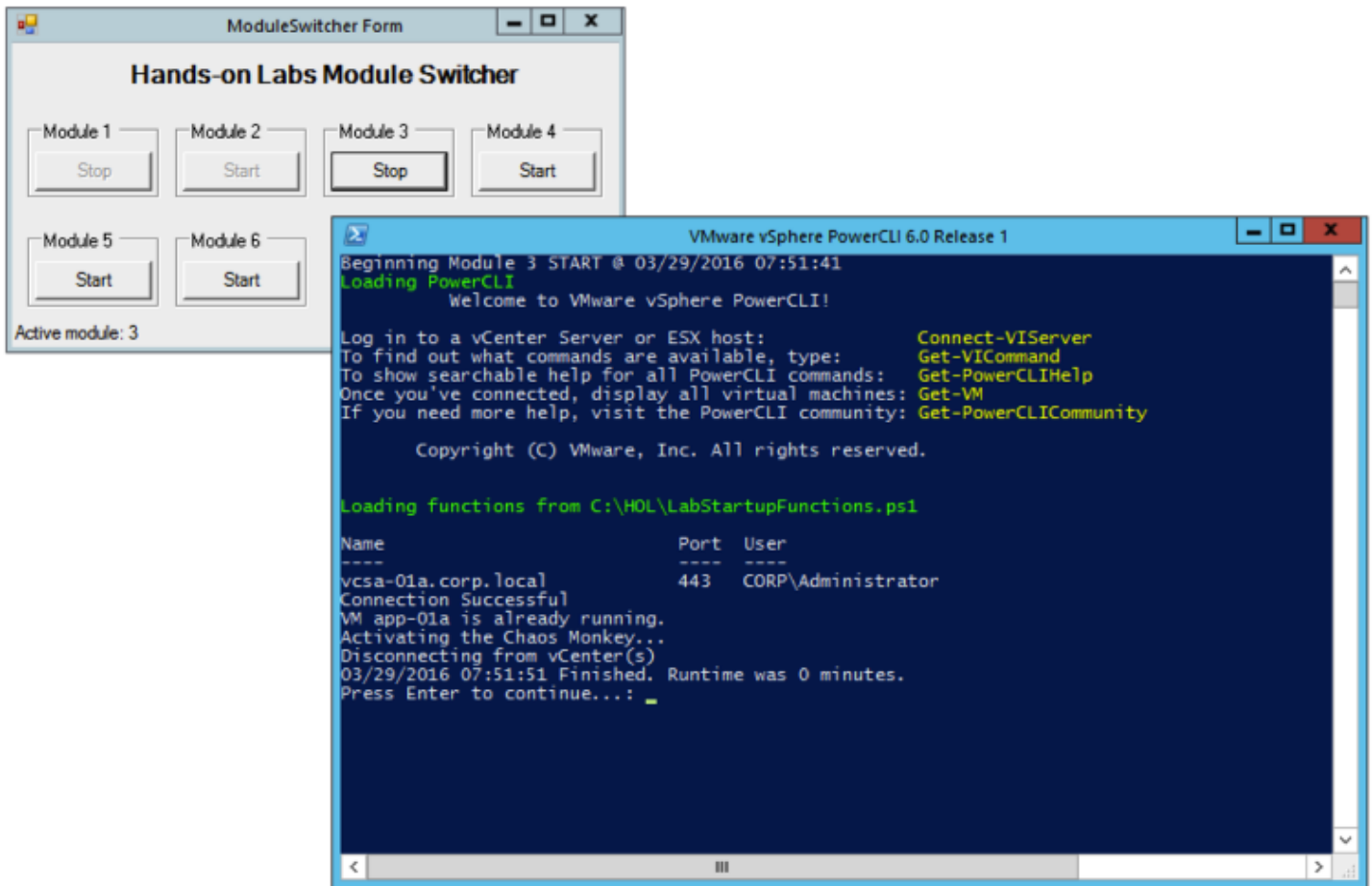
Continuing the example, and for the sake of completeness, jumping to Module 5 by clicking on its Start button will run the Module 3 script with the STOP action (since it is the current active module) and then run the Module 5 script with the START action. The buttons for Modules 1 through 4 are disabled because they have been completed or bypassed. Note that you can see which ones have been bypassed because the disabled buttons still show the name "Start"

OOPS! I closed the form/window/script...

Getting rid of the Module Switcher form is a matter of clicking the "X" in the red box in the upper right corner. The thought is that this form will remain open throughout the lab session, and the window can be minimized as needed. Things happen and in the event that the form is accidentally closed, it can be reopened by re-running the batch file and the state will be preserved: the current active module will be preserved and the previous buttons will be disabled.

Note: to preserve state, the currently active module number is stored in a text file called currentModule.txt inside the ModuleSwitcher directory. In the event that you want to test your ModuleSwitcher from the beginning, you can exit the form, delete this file, and then reopen the form. The ModuleSwitcher script will automatically ignore a state file that was not created on the current day. This prevents stale files within captured vPods from causing issues for users.

User Workflow



What does it look like when a user clicks the Start button? In this example, the user has clicked the Module 3 Start button from the initial lab state (active module = 1). In this image, the Module01 script has been run with the STOP action and the user has pressed ENTER to continue.

The Module03.ps1 script has been called with the START action in the new PowerShell window and is waiting for user confirmation.

In this case, the Module03 script needs to check and manage power state of a VM, so it loads PowerCLI and LabStartupFunctions.ps1, connects to the vCenter, and then checks the state of the app-01a VM. If that VM is powered off, this script will start it. Otherwise, it just reports that the VM is online, no warnings or errors, then activates the “Chaos Monkey” and waits for the user to press the ENTER key.

Known Limitations

This simple script has some known limitations which are called out in this section.

1. A Module##.ps1 script must exist for every module in the lab, even if the script does nothing more than answer to the START action and return. **Please be sure that you include a Module##.ps1 script for every module in your lab.**

2. A maximum of 20 modules supported (5 rows of 4 buttons) per instance an error message is thrown on the console if more than 20 Module##.ps1 scripts are detected. **Please keep the number of modules to 20 or fewer. Even at 20 buttons, the panel starts to consume a lot of screen real estate and look a bit cluttered. If you really need more, please contact the HOL core team.**
3. The title text for the window and the large “Hands-on Labs Module Switcher” text is a single line and a fixed width (the *static* width of the window). At this time, the text’s length is not validated. **Please keep the name of your ModuleSwitchDir folder to < 20 characters to prevent display issues.**
4. The LabStartupFunctions code has been developed with some specifics for LabStartup execution, including automatic remediation when failures are detected. While this should not typically cause issues in a pod that has reached the Ready state, we are in the process of validating and updating the contained functions as required. While it may seem obvious, it is worth noting that using these functions requires formatting the inputs the way that they are in the LabStartup.ps1 script.
5. Multiple simultaneous executions - the script does not handle being run multiple times simultaneously. Doing so will create multiple panels and may confuse the user.

Obtaining the ModuleSwitcher

The ModuleSwitcher code is not currently a part of the base vPod images. The script and its example files are available on an ISO image in the HOL-Dev-Resources catalog. Look for a media item called something like "ModuleSwitcher-v1.19" (note that the version number may be different if there have been changes).

The recommended location for the files is C:\HOL\ for the main ModuleSwitcher.ps1 script and C:\HOL\ModuleSwitcher for the Module##.ps1 scripts *if only one panel is needed in the vPod*. For multiple panels, as in a co-op vPod, the structure should look something like the following, where HOL-1700-1, HOL-1700-2 and "My Really Long Module Name" are different sets of modules/buttons corresponding to different manuals.

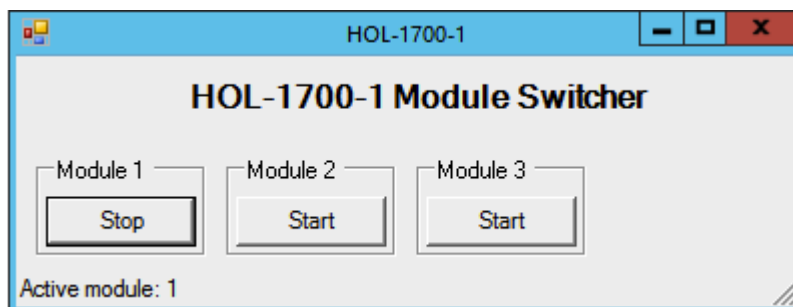
```
C:\HOL
|
|   ModuleSwitcher.ps1
|
|   └── ModuleSwitcher
|       |
|       |   MS_for_HOL-1700-1.bat
|       |   MS_for_HOL-1700-2.bat
|       |
|       |   └── HOL-1700-1
|           |
|           |   Module01.ps1
|           |   Module02.ps1
|           |   Module03.ps1
|           |   moduleMessages.txt
|           |
|           └── HOL-1700-2
|               |
|               |   Module01.ps1
|               |   Module02.ps1
|               |   Module03.ps1
```

```
Module04.ps1
Module05.ps1
Module06.ps1
Module07.ps1
Module08.ps1
Module09.ps1
Module10.ps1
Module11.ps1
Module12.ps1
Module13.ps1
Module14.ps1
Module15.ps1
Module16.ps1
Module17.ps1
My Really Long Module Name
Module01.ps1
Module02.ps1
```

The *MS_for_HOL-1700-1.bat* and *MS_for_HOL-1700-2.bat* files at the root of C:\HOL\ModuleSwitcher are batch files that launch the respective panels.

As the following article indicates, this structure can be used to produce 3 different panels for 3 different sets of modules.

Advanced Configuration



There are three advanced options that are implemented as command line parameters to the ModuleSwitcher.ps1 script.

-Force : this is a command line switch which ignores the data stored in currentModule.txt file, enables all buttons, sets the current active module to 1, and overwrites the data in currentModule.txt. This option is a shortcut during development and should not be used in a released pod since it defeats the state preservation function.

-ModuleSwitchDirPath : this option takes a string which is a custom path to the directory containing the ModuleXX.ps1 scripts. The name of the directory is also used in the name displayed on the ModuleSwitcher window. This means that it is possible to have multiple *sets* of module scripts which can be used depending on the manual the user is following.

-PanelName : this option allows a specific name to be used for both the panel's window title and the large text in the window. Do not use a title that is too long or the text will be "squished" or overflow the bounds of the window.

Calling

```
C:\HOL\ModuleSwitcher.ps1 -ModuleSwitchDirPath C:\HOL\ModuleSwitcher\HOL-1700-1
```

Results in the panel in this step.

Note that both the title of the window and the name displayed within the window reflect the name of the specified directory. The script uses the leaf directory name, so the path may include other intermediate directories if it facilitates organization.

Note: If the default *ModuleSwitcher* name is used, the panel will show *Hands-on Labs Module Switcher* instead of the folder name.

Example - Module 3

This is the example code for Module03.ps1. To keep the code cleaner, the START and STOP actions have each been encapsulated within their own functions: ModuleStart and ModuleStop. The main logic is simplified to:

```
if( $args[0] -eq 'START' ) {  
    ModuleStart  
} else {  
    ModuleStop  
}  
PAUSE
```

A LoadPowerCLI function has been written so that it may be reused in other module scripts. This also loads the *C:\HOL\LabStartupFunctions.ps1* file to access some of the pre-written code for connecting to vCenter. Note that a bare Connect-ViServer could be called here, but this shows how to access LabStartupFunctions if needed:

Module03.ps1

```
#  
# Module03.ps1 - Module 3 Start/Stop Script EXAMPLE  
#  
#Path to HOL directory in the vPod  
$holPath = 'C:\HOL'  
#This script's module number -- for output clarity  
$moduleNumber = 3
```

```

Function ModuleStart {
    $startTime = $(Get-Date)
    Write-Host "Beginning Module $moduleName START @ $startTime"

    LoadPowerCLI
    #Load LabStartupFunctions
    $InvocationPath = Join-Path $holPath 'LabStartupFunctions.ps1'
    if( Test-Path $InvocationPath ) {
        Write-Host -ForegroundColor Green "Loading functions from $InvocationPath"
        . $InvocationPath
    } else {
        Write-Host -ForegroundColor Red "ERROR: Unable to find $InvocationPath"
        Break
    }

    #TODO: Write your Module START code here
    $modVMs = ('app-01a:vcsa-01a.corp.local')

    Connect-VC 'vcsa-01a.corp.local' 'administrator@corp.local' 'VMware1!' ([REF]$result)
    foreach( $record in $modVMs ) {
        ($name,$vcenter) = $record.Split(":")
        $vm = Get-VM $name -server $vcenter
        if( $vm.PowerState -eq 'PoweredOn' ) {
            Write-Host "VM $name is already running."
        } else {
            Write-Host "VM $name is starting."
            Start-VM -VM $vm -Server $vcenter -Confirm:$false -RunAsync | Out-Null
            Sleep -Sec 10
        }
    }

    Write-Host "Activating the Chaos Monkey..."
    #You can do other things you need here, too
    Start-Sleep -Seconds 5

    Write-Host "Disconnecting from vCenter(s)"
    Disconnect-Viserver * -Confirm:$false

    Write-Host $( "$(Get-Date) Finished. Runtime was {0:N0} minutes." -f ((Get-RuntimeSeconds
    $startTime) / 60) )

} #ModuleStart
#####
Function ModuleStop {
    $startTime = $(Get-Date)
    Write-Host "Beginning Module $moduleName STOP @ $startTime"
    LoadPowerCLI
    #Load LabStartupFunctions
    $InvocationPath = Join-Path $holPath 'LabStartupFunctions.ps1'
    if( Test-Path $InvocationPath ) {

```

```

    Write-Host -ForegroundColor Green "Loading functions from $InvocationPath"
    . $InvocationPath
} else {
    Write-Host -ForegroundColor Red "ERROR: Unable to find $InvocationPath"
    Break
}

#TODO: Write your Module STOP code here

$modVMs = ('app-01a:vcasa-01a.corp.local')

Connect-VC 'vcasa-01a.corp.local' 'administrator@corp.local' 'VMware1!' ([REF]$result)
foreach( $record in $modVMs ) {
    ($name,$vcenter) = $record.Split(":")
    $vm = Get-VM $name -server $vcenter
    if( $vm.PowerState -eq 'PoweredOff' ) {
        Write-Host "VM $name is already powered off."
    } else {
        if( $vm.ExtensionData.Guest.ToolsRunningStatus -eq 'guestToolsRunning' ) {
            Write-Host "VM $name is shutting down."
            Stop-VMGuest -VM $vm -Server $vcenter -Confirm:$false | Out-Null
            Sleep -Sec 20
        } else {
            Write-Host "VM $name is being powered off."
            Stop-VM -VM $vm -server $vcenter -RunAsync -Confirm:$false | Out-Null
        }
    }
}
}

Write-Host "Deactivating the Chaos Monkey..."
#You can undo other things you need here, too
Start-Sleep -Seconds 5

Write-Host "Disconnecting from vCenter(s)"
Disconnect-Viserver * -Confirm:$false
Write-Host "$(Get-Date) Finished. Runtime was {0:N0} minutes." -f ((Get-RuntimeSeconds
$startTime) / 60) )
} #ModuleStop
#####
# supporting functions
Function LoadPowerCLI {
    Try {
        #For PowerCLI v6.x
        $PowerCliInit = 'C:\Program Files (x86)\VMware\Infrastructure\vSphere PowerCLI\Scripts\
Initialize-PowerCLIEnvironment.ps1'
        Write-Host -ForegroundColor Green "Loading PowerCLI"
        . $PowerCliInit
    }
    Catch {
        Write-Host -ForegroundColor Red "No PowerCLI found, unable to continue."
        Break
    }
}

```

```

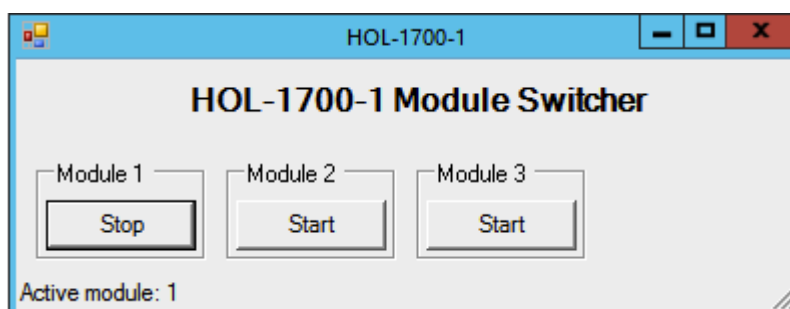
    }
} #LoadPowerCLI
#####
# The main program logic
if( $args[0] -eq 'START' ) {
    ModuleStart
} else {
    ModuleStop
}
}
#Wait so that the user can see any messages and control the exit
PAUSE

```

More Example Code - the **EXAMPLE** directory

The Module Switcher package includes an *EXAMPLE* directory which contains two separate module directories and two batch files which can be used to launch panels from those directories.

HOL-1700-1

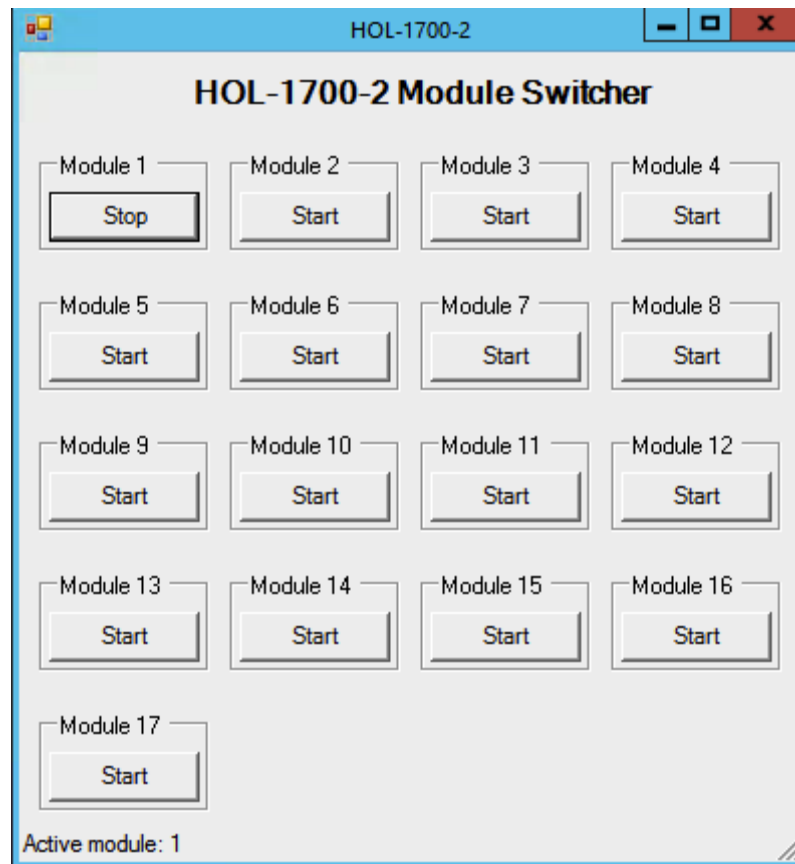


This example contains 3 modules. Module01.ps1 is a skeleton that simply prints out that it is stopping. The START action for Module 1 is never called explicitly as it is assumed that LabStartup handles getting the environment to Module 1 state.

Module02.ps1 is an example that loads PowerCLI and LabStartupFunctions, then performs action on some vVMs. Note that this will not function correctly if the vVMs are not present in your environment, but you can modify the script to reference vVMs in your pod. Finally, Module03.ps1 is more of a shell or starting point which contains minimal suggested functionality.

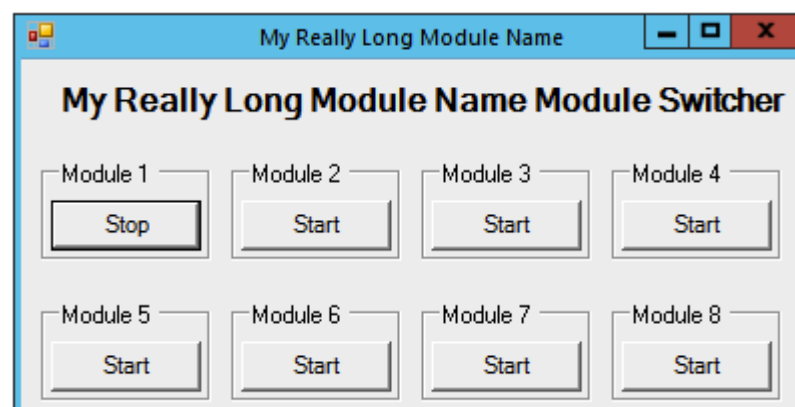
This is probably the code that you want to start with when building your new module scripts.

HOL-1700-2



This example has 17 modules. Each script is just a skeleton that responds to START/STOP action and reports which module's script was called. In addition to showing what the panel looks like with a large number of buttons and how to load a different set of modules using the command line options, this can be used to test the user workflow and script-calling logic of the ModuleSwitcher.

My Really Long Module Name



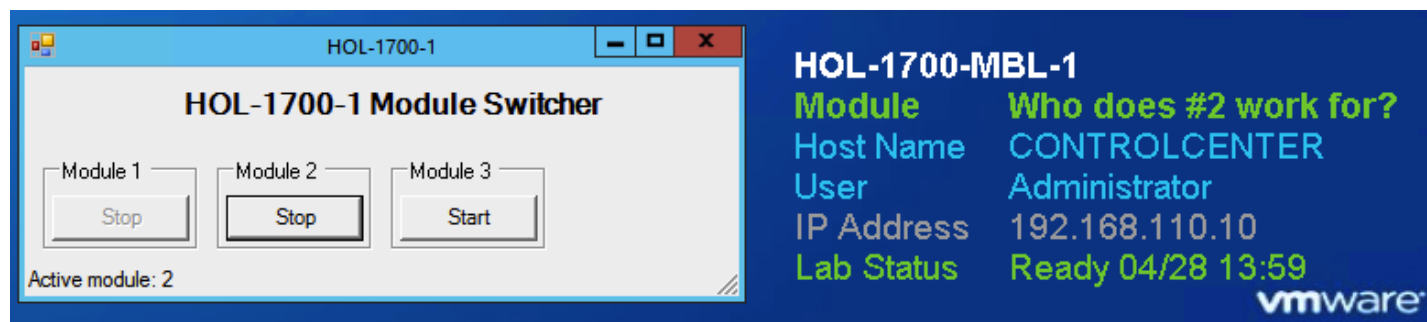
Because the Module Switcher script will take the name of the ModuleSwitchDirPath directory as the window title and form header, you can place text that you want in those locations. Be aware, though,

that the text box is a fixed size, so the amount of text you can put in there is limited. Too much text will overrun the box and/or cause it to look cramped.

By default, the text " Module Switcher" is appended to your text to create the form header text.

If you want to use something else for the header text, specify that string on the command line using the *-PanelName* option.

How To: Display Current Module on DesktopInfo



Some of the feedback we received on the initial version of the Module Switcher was that it would be nice to display the currently active module on the desktop in addition to inside the ModuleSwitcher window. This is accomplished by an edit to the *C:\DesktopInfo\DesktopInfo.ini* file to provide a spot for the data, a file containing one line of "message" data to display, and a couple of lines to reset its contents at Main Console shutdown:

1. C:\DesktopInfo\DesktopInfo.ini (add around line 25, in the [items] section after the COMMENT):

```
#ModuleSwitcher
FILE=active:1,interval:20,color:55CC77,style:b,type:text,text:Module,file:C:\HOL\ModuleSwitcher\
currentMessage.txt
```

2. <ModuleSwitchDirPath>\moduleMessages.txt:

```
The loneliest number
Who does #2 work for?
Three's a crowd
I am Number Four
Number Five is Alive!
Six Ways to Sunday
The Magnificent Seven
```

This file is provided per panel by including it in the ModuleSwitchDirPath (with the Module##.ps1 files). Each line in the file corresponds to one module: line 1 is module 1, line 2 is module 2, and so on. Using the above file, ModuleSwitcher wipp populate the *C:\HOL\ModuleSwitcher*

currentMessage.txt file with the corresponding data. So, a few seconds after module 2 becomes the active module, the desktop will show as in the image associated with this step.

NOTE: There is limited space available in the DesktopInfo region, 20-25 characters, so please keep the text you wish to display as short as possible while still being useful.

3. C:\HOL\LabLogoff.ps1

Unless you want to remember to cleanup the file each time you shut down your pod, one of the automatic scripts should be modified to reset the *currentMessage.txt* file. If this step is not performed and the pod was shut down while Module 4 was active, it will boot up showing Module 4 as the active module until the user clicks on another module in the Module Switcher. To address this, add these two lines to the end of the LabLogoff script and let the system handle it for you.

```
$messageFile = 'C:\HOL\ModuleSwitcher\currentMessage.txt'
Set-Content -Value "None Selected" -Path $messageFile
```