

# Windows Scripts

## Batch vs PowerShell

PowerShell scripts are newer and offer greater functionality than traditional batch files. PowerShell also has a development environment (PowerShell ISE), which makes writing and testing scripts easier:

<https://msdn.microsoft.com/en-us/powershell/scripting/core-powershell/ise/introducing-the-windows-powershell-ise>

We still use batch files when calling them from Max, since calling PowerShell scripts requires a number of untested work-arounds.

## Startup Scripts

On each of our exhibits, Task Scheduler is set up to start a process at logon. On a machine that runs only one application, that application itself can be the startup process. With the transition to Windows 10, a startup script is probably going to be the preferred option, even when simply launching one application, because scripting allows us to deal with the Taskbar-in-front problem.

Here is a canonical startup script. It launches two programs (CoolSynthCV and CoolSynth) and ensures that the second one is full-screen:

```
#allow five seconds for user to cancel script
"Cool Synth starting up in five seconds. Close this window to cancel..."
Start-Sleep -s 1
"4..."
Start-Sleep -s 1
"3..."
Start-Sleep -s 1
"2..."
Start-Sleep -s 1
"1..."
Start-Sleep -s 1
""

"Please wait, Cool Synth startup in progress, this window will close when finished."

#run the Cool Synth CV application
Start-Process -FilePath "D:\dev\CoolSynthCV\CoolSynthCV.exe"
$CV = Get-Process CoolSynthCV
$CV.ProcessorAffinity = 192

#pause while CV app starts
Start-Sleep -s 30
#make CV app active to clear taskbar notification
Set-WindowActive -Window $CV.MainWindowHandle
Start-Sleep 2
```

```
#run the CoolSynth application
Start-Process -FilePath "D:dev\CoolSynth\CoolSynth.exe"
$RR = Get-Process CoolSynth
$RR.ProcessorAffinity = 63

#take focus, minimize and then maximize RR to prevent anything from showing in front of it
Start-Sleep 4
Set-WindowActive -Window $RR.MainWindowHandle
Start-Sleep 2
Set-WindowPosition -Minimize -Window $RR.MainWindowHandle
Start-Sleep 2
Set-WindowPosition -Maximize -Window $RR.MainWindowHandle
```

The above was included as text, so that it can be copy/pasted.

Now Let's look at each section of the script.

```
#allow five seconds for user to cancel script  
"CoolSynth starting up in five seconds. Close this window to cancel..."
```

Lines preceded by # are comments, use them to remind your future self of what the script is doing. The second line prints a message, telling the user what is about to happen. In PowerShell, just place something in quotes and it will be written to the standard output.

```
Start-Sleep -s 1  
"4..."  
Start-Sleep -s 1  
"3..."  
Start-Sleep -s 1  
"2..."  
Start-Sleep -s 1  
"1..."  
Start-Sleep -s 1  
"  
"Please wait, CoolSynth startup in progress, this window will close when finished."
```

Above you see the command `Start-Sleep`, which takes the parameter `-s` and causes the thread to sleep for the specified number of seconds. In this case it sleeps for one second at a time, counting down five seconds and writing to console so that the user can see the countdown. Another message follows, informing the user that application startup has begun.

```
Start-Process -FilePath "D:\CoolSynthCV.exe"
```

`Start-Process` takes a `-FilePath` argument to an executable to start. In this case, the CoolSynthCV.exe application is being launched.

```
$CV = Get-Process CoolSynthCV
```

In PowerShell, symbols that begin with `$` are variables. This line uses the `Get-Process` command to return the Process ID for the newly launched CoolSynthCV application, and stores it in the variable `$CV`. A process ID is the number the Windows scheduler actually uses to identify processes running on the computer.

```
$CV.ProcessorAffinity = 192
```

This command sets the Processor Affinity for the CoolSynthCVPID stored in `$CV`. Processor Affinity describes which logical cores (real or virtual) a process runs on. In the case of CoolSynth we are going to be launching two applications which have been demonstrated to perform better when manually assigned to different physical processor cores. The assignment is a bitmask, the length of which depends upon how many cores the CPU has. In the case of CoolSynth, which runs on a four core CPU with Hyper-Threading (two logical cores per physical core), this makes for 8 logical cores.

Here is a breakdown of how the mask works:

**Processor Affinity Bitmask Table: (8 Core Processor)**

Core # = Value = BitMask

Core 1 = 1 = 00000001

Core 2 = 2 = 00000010

Core 3 = 4 = 00000100

Core 4 = 8 = 00001000

Core 5 = 16 = 00010000

Core 6 = 32 = 00100000

Core 7 = 64 = 01000000

Core 8 = 128 = 10000000

Add the decimal values together to use multiple cores.  $255 = 11111111$  = all 8 cores.

The command `$CV.ProcessorAffinity = 192`, is assigning the CoolSynthCV application to cores 7 and 8 ( $128 + 64 = 192$ ). These two logical cores correspond to physical core 4 on the CPU.

```
#pause while CV app starts
Start-Sleep -s 30
```

Sleep for 30 seconds, to allow the application CoolSynthCV to fully start, before launching other applications.

```
#make CV app active to clear taskbar notification
Set-WindowActive -Window $CV.MainWindowHandle
Start-Sleep 2
```

`Set-WindowActive` gives a window command focus. This is useful, because giving an application focus clears the glowing notification that keeps the Taskbar in front. Essentially, the notification says "hey, this application is open!" To which, we reply "thanks, I see it." The `-Window` parameter takes a `MainWindowHandle` argument. We return the `MainWindowHandle` from the PID stored in `$CV`, with `$CV.MainWindowHandle` and use it as the argument to `-Window`.

```
#run the CoolSynth application
Start-Process -FilePath "D:\dev\CoolSynth\CoolSynth.exe"
$RR = Get-Process CoolSynth
$RR.ProcessorAffinity = 63
```

As with CoolSynthCV, we launch the main CoolSynth application with `Start-Process`, store the PID in a variable, and set the processor affinity ( $1 + 2 + 4 + 8 + 16 + 32 = 63$ ). We are assigning CoolSynth to logical cores 1-6, corresponding to physical cores 1-3.

```
#take focus, minimize and then maximize CS to prevent anything from showing in front of it
Start-Sleep 4
Set-WindowActive -Window $RR.MainWindowHandle
Start-Sleep 2
```

We only take 4 seconds waiting for CoolSynth to start up, whereas we waited for 30 seconds with CoolSynthCV. This is partly because CoolSynth takes longer to start (mainly due to the Max runtime starting up), but also because nothing will start up that depends on CoolSynth, so we don't need to be as careful to make sure it is fully up and running before moving on. We then activate CoolSynth's main window, to clear the Taskbar notification.

```
Set-WindowPosition -Minimize -Window $RR.MainWindowHandle  
Start-Sleep 2  
Set-WindowPosition -Maximize -Window $RR.MainWindowHandle
```

Just to be safe, we take one final step to ensure that CoolSynth is maximized and in front of everything else that may have popped up along the way. We do this by passing the MainWindowHandle from the PID in \$RR with the `-Minimize` command, followed by passing it again with the `-Maximize` command, putting us into full screen as the last step in starting the exhibit.

This method of minimize/maximize is not universally proven to work. On one installation, we instead used the `Set-WindowActive` message on a number of applications in succession, to clear all of their taskbar notifications before maximizing the full screen application. So far though, some combination of these methods has gotten us the desired result with Windows 10.

## PowerShell Snap-ins

Powershell commands can easily be extended with libraries, called Snap-ins. In this startup script the commands `Set-WindowActive` and `Set-WindowPosition` are from the Windows Automation Snap-in for Powershell (WASP):

<https://wasp.codeplex.com/>

I prefer placing Snap-ins in:

```
C:\Users\AP\Documents\WindowsPowerShell\Modules
```

This is in the default path for PowerShell, so your scripts can make use of them there. This also makes them easy to find/copy/document when development on an exhibit is finished.

## Shutdown and Restart Scripts

If you are scheduling shutdown or restart, you can just schedule the restart or shutdown directly in Task Scheduler. However, if you want to have Max trigger shutdown or restart, it can do so by creating a batch file with either of these lines:

```
shutdown.exe /s  
shutdown.exe /r
```

Restart and shutdown are both performed by shutdown.exe, the difference is whether you pass the `/s` flag (for shutdown) or the `/r` flag (for restart). Place either one in a batch file and Max can run it with `;max launchbrowser`.

# Scheduling Startup Script

## Security Policy

By default, execution of PowerShell scripts is restricted. Unless a user runs them manually, they must be signed by a trusted authority. This security policy can be changed:

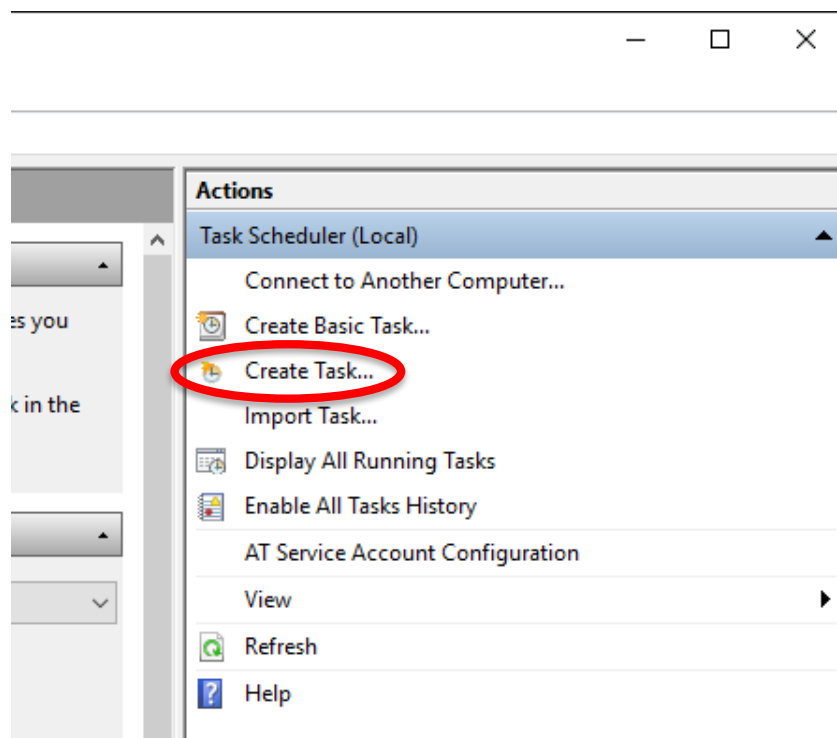
<https://technet.microsoft.com/en-us/library/ee176961.aspx>

...but there are safer ways to deal with this for our purposes. If you look at the detailed Windows 10 Deployment Checklist document, you will see that the startup task does not launch the startup script directly. Instead, it launches PowerShell.exe and passes arguments, including the name of the startup script and the `-executionpolicy bypass` argument. This tells Windows to allow this particular script to run as an exception to the security policy on scripting.

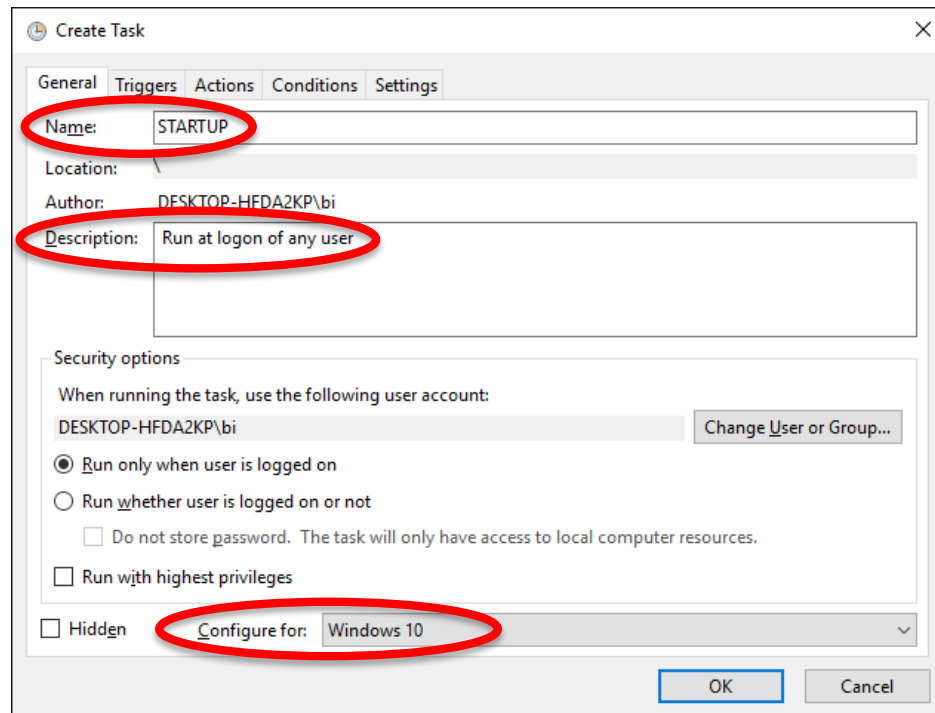
## Task Scheduler

To schedule a PowerShell script to run at logon, follow these steps:

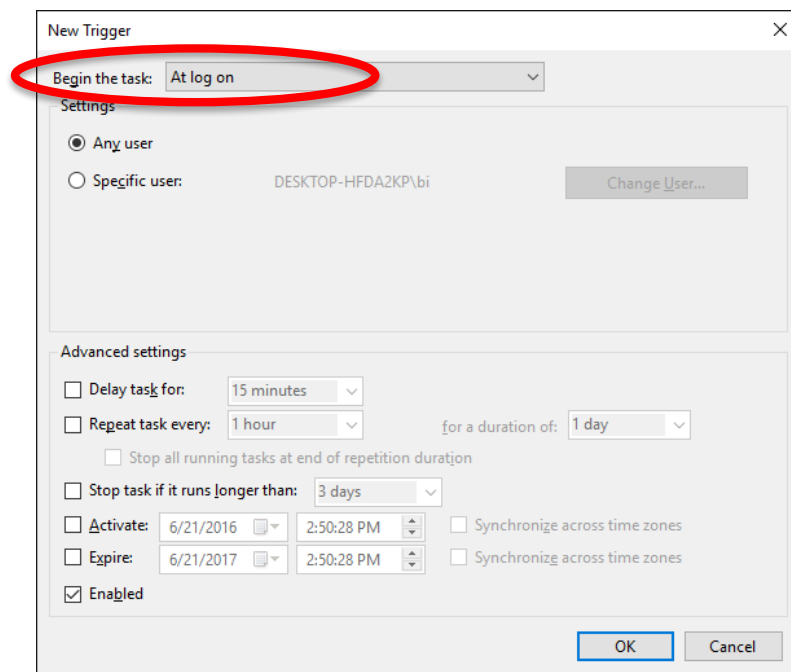
- *Start* > “Task Scheduler” > *Create Task...*



- In the *General* tab, give the task a name and description, set *Configure for Windows 10*.



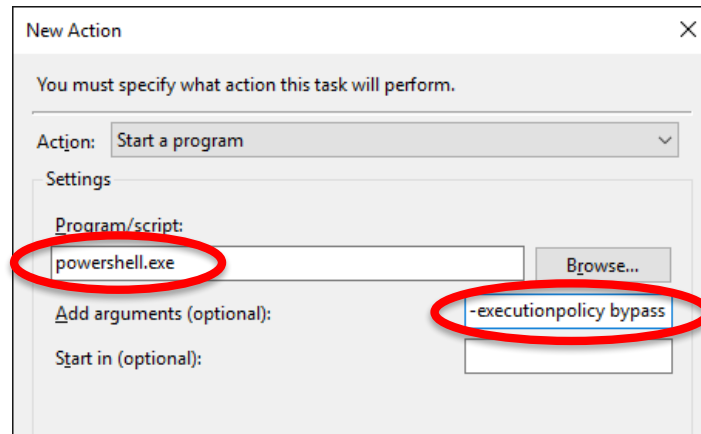
- In the *Triggers* tab, click *New* and then set *Begin the task* to “At log on”



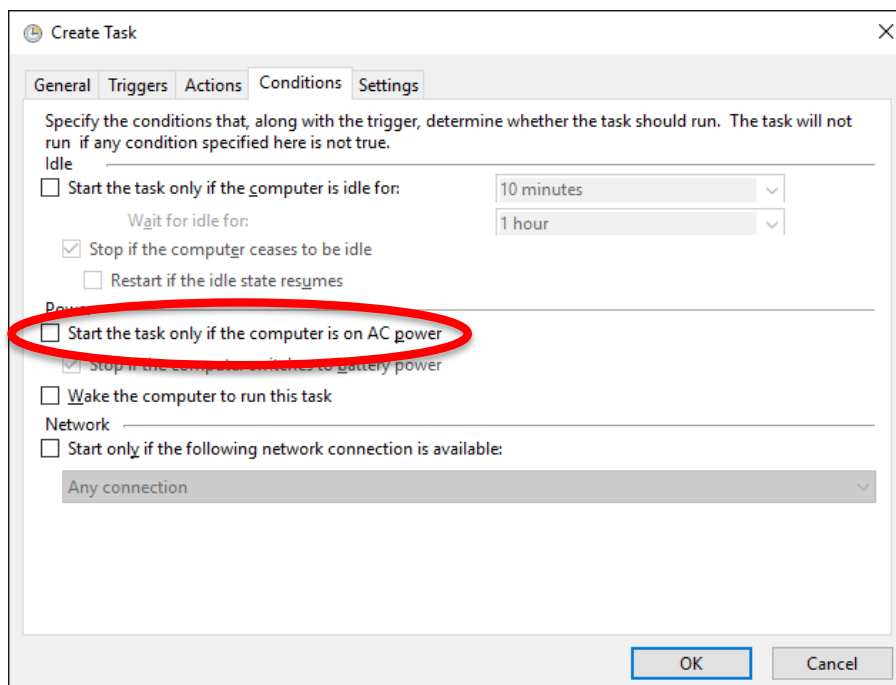
- In the *Actions* tab, click *New*. Enter “powershell.exe” in *Program/script* and put the following in *Add arguments*:

-executionpolicy bypass -file C:\path\to\script

- The first argument tells the computer that this script should not be blocked, even if there is a security policy disallowing scripts.
- The second argument tells PowerShell which script to run.
- The whole argument list is too long to be previewed below, but both arguments are in there.



- If this is a laptop or tablet, which may be started while on battery power, go to the *Conditions* tab and uncheck *Start the task only if the computer is on AC power*.



- Click *OK*