Project Documentation  
Peripheral Sleep Config Bandaid

# Table of Contents

[Table of Contents 1](#_Toc165402211)

[The Problem 4](#_Toc165402212)

[Problem Background 4](#_Toc165402213)

[Problem description 4](#_Toc165402214)

[Hardware architecture 4](#_Toc165402215)

[Problem assumptions (Safe assumptions) 4](#_Toc165402216)

[Core Problem Being Addressed 5](#_Toc165402217)

[Project Deadline 5](#_Toc165402218)

[Proposed Solution 5](#_Toc165402219)

[Solution Hypothesis 5](#_Toc165402220)

[Solution Description 5](#_Toc165402221)

[Solution Features 6](#_Toc165402222)

[Project Scope 6](#_Toc165402223)

[In Scope 6](#_Toc165402224)

[Out of Scope 6](#_Toc165402225)

[Project Requirements 6](#_Toc165402226)

[Internal Requirements 6](#_Toc165402227)

[External Requirements 6](#_Toc165402228)

[Project Roadmap 7](#_Toc165402229)

[Development 8](#_Toc165402230)

[Environment 8](#_Toc165402231)

[Notes/Snapshots 8](#_Toc165402232)

[Snapshot [Date] - [Topic] - Comments 8](#_Toc165402233)

[Issues 8](#_Toc165402234)

[Standards 8](#_Toc165402235)

[Naming Conventions 8](#_Toc165402236)

[Spacing standards 9](#_Toc165402237)

[Contribution Standards 9](#_Toc165402238)

[Testing Standards 9](#_Toc165402239)

[Documentation Standards 9](#_Toc165402240)

[Quality Assurance 9](#_Toc165402241)

[Quality Assurance Plan 9](#_Toc165402242)

[Testing Snapshots 9](#_Toc165402243)

[Deployment 9](#_Toc165402244)

[Deployment Plan 9](#_Toc165402245)

[Deployment Retrospective 9](#_Toc165402246)

[Project Learnings and Post-Mortem 10](#_Toc165402247)

[Things Learned for Development 10](#_Toc165402248)

[Things Learned During Development 10](#_Toc165402249)

[Things Learned During Implementation or from Feedback 10](#_Toc165402250)

[Information References 11](#_Toc165402251)

# The Problem

## Problem Background

### Problem description

1. The technicians have reported issues with devices that are plugged into power. Some workstations are having network cards or USB peripherals stop while devices are asleep. Identify changes that can be made to keep USB/network cards awake while devices are asleep while they are plugged into power. Write a PowerShell script to detect the current settings and output if the settings are correctly configured. The PowerShell script will be used as a compliance check. The script should output as true or false.
2. Write a second PowerShell script to remediate the issue when the power settings do not align with the defined detection settings used to remedy network cards and USB peripherals stopping while asleep.

### Hardware architecture

Devices in most computers are connected to an architecture that goes from the board to the PCIe buses, to the USB root hubs, to the end device and any of its adjacent components. A known solution to this issue is to disable sleep on the root hubs since some devices will encounter errors when ghost power or their own power continues their communication attempts with the host pc which induces an error within their own internal controllers which are then presented as a device error to the end user rendering the device unusable.

## Problem assumptions (Safe assumptions)

All systems being addressed are Windows based systems due to the use of Powershell. Script may be used by a less technical person, meaning that the script needs to be a single document, or the probability of user increases exponentially.

Most systems within the fleet will be running a version of Powershell that still has the WMI command interface. But this can be validated with this command:

A screenshot of a computer

Description automatically generated

Note, that this command was run on a machine running Windows 11 with PowerShell 5.1 but there are higher versions of PowerShell available. These higher versions of PowerShell have deprecated the use WMI and may not support the commands within the solution. Need to include a contingency for that edge case.

## Core Problem Being Addressed

The primary issue being addressed is a Windows Modern Standby issue since not all devices are modern standby certified.

Secondary problem that will also be addressed is a user/driver install error issue where devices where disabled but not re-enabled.

## Project Deadline

Per the constraints of the client, the deadline of this project is 5/6/2024

# Proposed Solution

## Solution Hypothesis

All USB peripherals are connected to USB root hubs at some point between the end component and the PCIe component. PCIe components cannot be configured to keep power enabled. The USB root hubs on the other hand can be configured to have power remain on when the system is asleep.

Another potential problem is that the device may simply have been disabled by user error or an update (I’m looking at you Intel with your recent WIFI update)

## Solution Description

The first step will be to check the version of PowerShell to ensure script compatibility.

Then, we will get a list of all the USB class devices on the assumption that by disabling sleep control on the hubs, all the problems will go away. This task can be made faster by filtering for devices that have sleep enabled.

Then we will go through each of those devices and then disable sleep, WMI may have a faster way to do this but the easiest solution to implement would be a simple loop. Will consider alternative solutions if there is time.

Then we will get the list of network adapters that are disabled, and then enable all of them. This will have a side effect of also enabling any problematic devices with known issues that haven’t been completely replaced if there is an ongoing issue with them, so a notice will need to be posted for those types of situations.

## Solution Features

* Script that will verify if connected devices are configured correctly
  + Check power control status of individual USB hub devices
  + Checks the enable/disable status of USB peripheral devices
  + Checks the enable/disable status of network adapters and physical wifi devices
* Script that will adjust device configurations to eliminate the device sleep issue
  + Adjusts the power control status to disable sleep control on USB hub devices
  + Enable all installed USB peripheral devices
  + Enable all installed network adapters that are not virtual adapters

# Project Scope

## In Scope

* Identifying USB devices
* Identifying network adapters
* Checking power control status of USB hub devices
* Checking status of installed USB peripheral devices
* Checking status of installed network adapters and physical wifi devices

## Out of Scope

* Plugging in the device

# Project Requirements

## Internal Requirements

## External Requirements

# Project Roadmap

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Status | Milestone | Blocks No. |
| 0 | In Progress | Initial exploration | 1.0, 2.0 |
| 1.0 | TODO | Create a script that will list out all USB class devices and Network adapters | 1.1 |
| 1.1 | TODO | Modify the script from 1.0 to display the power configurations of the devices listed | 1.2 |
| 1.2 | TODO | Modify the script from 1.1 to create another list of the devices that do not match the expected power configuration | 1.3 |
| 1.3 | TODO | Output false if the list from 1.2 is greater than 0 which will indicate a compliance failure | 3.0 |
| 2.0 | TODO | Write a script that creates a list of all USB hub devices and network adapters that fail the compliance check based on filters like the ones found in 1.2 | 2.1 |
| 2.1 | TODO | Step through the lists and output the names and status of the devices | 2.2, 2.3, 2.4 |
| 2.2 | TODO | Modify the script to enable USB power policy modification to bring the hubs into compliance | 3.0 |
| 2.3 | TODO | Modify the script to enable the USB devices that may be disabled but still installed | 3.0 |
| 2.4 | TODO | Modify the script to enable any disabled network adapters that are not virtual adapters | 3.0 |
| 3.0 | TODO | Integration testing on at least 4 unique devices | 4.0 |
| 4.0 | TODO | Submission to client | 4.0 |

# Development

## Environment

Operating system: Windows

IDE: Notepad++ or VSCode

Language: Powershell

## Notes/Snapshots

### Snapshot 2024-05-04 - Finding USB Devices - Comments

Potential direction:

https://learn.microsoft.com/en-us/powershell/module/pnpdevice/get-pnpdevice?view=windowsserver2022-ps  
A screenshot of a computer screen

Description automatically generated

There might be a gwmi command that we could use but we need to identify the properties.

<https://devblogs.microsoft.com/powershell/displaying-usb-devices-using-wmi/>

A computer screen shot of a blue screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Seems like we’re being brought back to get-wmiobject since the other options are not available

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-wmiobject?view=powershell-5.1>

Needed to use some select queries, but wasn’t 100% sure on the syntax.

<https://learn.microsoft.com/en-us/windows/win32/wmisdk/select-statement-for-data-queries>

So now the next step is figuring out which class it is that I need to use

<https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-classes>

A screenshot of a computer program

Description automatically generated

The property that needs to be altered is the PowerManagementSupported property but that property is read-only meaning that there is some other channel through which this setting is adjusted. There may be a possibility that it is related to the registry keys for the device but that isn’t something that can be said with 100% confidence. This will have to be picked back up another day.

### Snapshot 2024-05-05 - Finding USB Devices - Comments

<https://superuser.com/questions/1075067/how-can-i-disable-power-saving-for-all-devices-in-device-manager>

using the suggestion found here, I could use Get-CimInstance and Set-CimInstance to control the power management feature of devices. With the first command as suggested from

A screenshot of a computer

Description automatically generated

Playing off of that, and realizing that the instance ids from Get-PNPDevice are the same as Get-CimInstance, I decided to filter down to a command that looks only at the devices that I am seeking

A screen shot of a computer

Description automatically generated

Get-CimInstance -ClassName MSPower\_DeviceEnable -namespace root\wmi | where {$\_.InstanceName -like "\*USB\*"}

The above command will get every single USB device that is handled under MSPower\_DeviceEnable which enables windows modern standby on these devices.

Learning a bit more about get-CimInstance

<https://learn.microsoft.com/en-us/powershell/module/cimcmdlets/get-ciminstance?view=powershell-7.4>

Learning about PowerShell where-object to filter the list down to specific devices

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/where-object?view=powershell-7.4>

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_regular_expressions?view=powershell-7.4>

The finalized command to get all of the potential devices that may be impacted by Windows Modern Standby are retrieved with the following command:

Get-PnpDevice | where {$\_.FriendlyName -Match "(Wi-Fi)|(Ethernet)|(Hub)"} | where {$\_.FriendlyName -notmatch "Virtual"} | where {$\_.Class -notmatch "Software"}

A screenshot of a computer

Description automatically generated

Adjusting the query to rename the instance id to instance name in order to engage in some create joinery

Research to double check that the feature is available since PowerShell seems to be deriving some operations from SQL

<https://devblogs.microsoft.com/powershell/join-object/>

A screen shot of a computer

Description automatically generated

The two commands that we need now

A screen shot of a computer

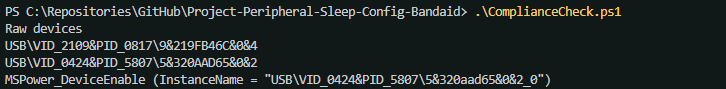
Description automatically generated

Next is to go through the list of PNP devices that match our criteria for devices covered and then add those device ids to an array and then filter our Get-CimInstance list based off of that array. Unfortunately, we cannot directly use the Get-PNPDevice output even if we have it filtered down to a single column.

Get-PnpDevice | where {$\_.FriendlyName -Match "(Wi-Fi)|(Ethernet)|(Hub)"} | where {$\_.FriendlyName -notmatch "Virtual"} | where {$\_.Class -notmatch "Software"} | select -property @{ label="InstanceName"; expression={$\_.InstanceID}; }

Get-CimInstance -ClassName MSPower\_DeviceEnable -namespace root\wmi

### Snapshot 2024-05-07 - Finding USB Devices - Comments



Realized that the instance names are not an exact match for the instance ids. In order to get around this problem, I used a regex match after a bit of string manipulation to make the instance id regex compatible to be able to find matches.

With this, we could build a filtered list of all the target devices that need to be checked and/or have their configuration fixed. This list can then be used to validate whether or not the devices are in compliance and correct said devices. This could have been wrapped into a single script, but due to the nature of the request, the script was made in two versions with the only difference being the inclusion of the Set-CimInstance command on line 34 of the peripheral fix script.

# Project Learnings and Post-Mortem

## Things Learned for Development

* The nuances of wmi objects and how they are not always directly able to interact with individual devices without additional wrappers like set-ciminstance.
* Ciminstance commands for controlling whether or not modern standby is enabled for individual devices that have the feature available.
* Get-pnpdevices for identifying devices that may be potential targets for the compliance check

## Things Learned During Development

* Cim Instance instance names are not the same as pnp device instance ids. This was a source of some headaches, since it wasn’t easy to directly identify whether or not there were direct matches without the use of notepad++ with the search feature.

A computer screen shot of a program

Description automatically generated

## Things Learned During Implementation or from Feedback

No feedback yet (TBD)

# Information References

Verifying that WMI can control devices:

<https://stackoverflow.com/questions/31688752/editing-device-manager-using-powershell>

Command to get a list of potential target devices (potential part of solution):

<https://learn.microsoft.com/en-us/powershell/module/pnpdevice/get-pnpdevice?view=windowsserver2022-ps>

Using Get-WMIObject

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-wmiobject?view=powershell-5.1>

Maybe check event log to see if a device was removed? This might have much more overhead both in terms of implementation time and execution time.

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-eventlog?view=powershell-5.1>