Project Documentation  
Peripheral Sleep Config Bandaid

# Table of Contents

[Table of Contents 1](#_Toc165315130)

[The Problem 4](#_Toc165315131)

[Problem Background 4](#_Toc165315132)

[Problem description 4](#_Toc165315133)

[Hardware architecture 4](#_Toc165315134)

[Problem assumptions (Safe assumptions) 4](#_Toc165315135)

[Core Problem Being Addressed 5](#_Toc165315136)

[Proposed Solution 5](#_Toc165315137)

[Solution Hypothesis 5](#_Toc165315138)

[Solution Description 5](#_Toc165315139)

[Solution Features 6](#_Toc165315140)

[Project Scope 6](#_Toc165315141)

[In Scope 6](#_Toc165315142)

[Out of Scope 6](#_Toc165315143)

[Project Requirements 6](#_Toc165315144)

[Internal Requirements 6](#_Toc165315145)

[External Requirements 6](#_Toc165315146)

[Project Roadmap 6](#_Toc165315147)

[Development 6](#_Toc165315148)

[Environment 6](#_Toc165315149)

[Notes/Snapshots 7](#_Toc165315150)

[Snapshot [Date] - [Topic] - Comments 7](#_Toc165315151)

[Issues 7](#_Toc165315152)

[Standards 7](#_Toc165315153)

[Naming Conventions 7](#_Toc165315154)

[Spacing standards 7](#_Toc165315155)

[Contribution Standards 7](#_Toc165315156)

[Testing Standards 7](#_Toc165315157)

[Documentation Standards 7](#_Toc165315158)

[Quality Assurance 8](#_Toc165315159)

[Quality Assurance Plan 8](#_Toc165315160)

[Testing Snapshots 8](#_Toc165315161)

[Deployment 8](#_Toc165315162)

[Deployment Plan 8](#_Toc165315163)

[Deployment Retrospective 8](#_Toc165315164)

[Project Learnings and Post-Mortem 8](#_Toc165315165)

[Things Learned for Development 8](#_Toc165315166)

[Things Learned During Development 8](#_Toc165315167)

[Things Learned During Implementation or from Feedback 8](#_Toc165315168)

[Information References 9](#_Toc165315169)

# 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.

# 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

# Development

## Environment

## Notes/Snapshots

### Snapshot [Date] - [Topic] - Comments

## Issues

|  |  |  |
| --- | --- | --- |
| Problem: | Problem Description/Notes: | Resolution: |
|  |  |  |
|  |  |  |

## Standards

### Naming Conventions

### Spacing standards

### Contribution Standards

### Testing Standards

### Documentation Standards

# Quality Assurance

## Quality Assurance Plan

## Testing Snapshots

# Deployment

## Deployment Plan

## Deployment Retrospective

# Project Learnings and Post-Mortem

## Things Learned for Development

## Things Learned During Development

## Things Learned During Implementation or from Feedback

# 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>