Application Notes

Recovering a USB 3.x xHCI Host Controller on Windows

Programmatically restarting a USB 3.x host controller without rebooting the PC

THE IMAGINGSOURCE
TECHNOLOGY BASED ON STANDARDS

Contents

1. Introduction

Industrial vision applications often require continuous and unattended operation. A temporary failure of the USB subsystem can therefore be problematic, particularly when a USB 3.x camera becomes inaccessible while the rest of the application and operating system continue to operate normally.

One possible cause is an error condition of the USB 3.x eXtensible Host Controller Interface (xHCI) controller.

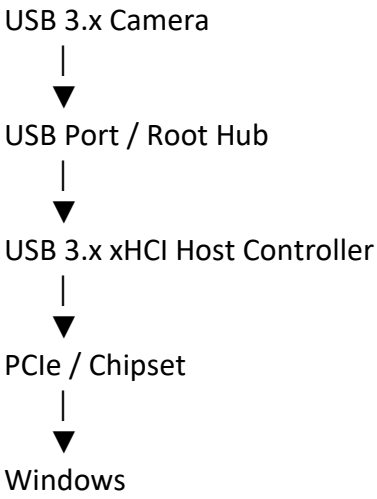
In such a situation, physically disconnecting and reconnecting the camera may not resolve the problem. Similarly, restarting the camera application may not be sufficient if the USB host controller itself remains in an error state.

Windows provides several mechanisms that can be used to restart or reinitialize a Plug and Play device without rebooting the entire operating system.

This Application Note describes several approaches, with a focus on Microsoft’s built-in PnPUtil utility and programmatic device-state management.

2. Problem Description

A typical failure scenario can look like this:



If the xHCI controller enters an unexpected state, the camera may no longer be accessible to the application.

Possible symptoms include:

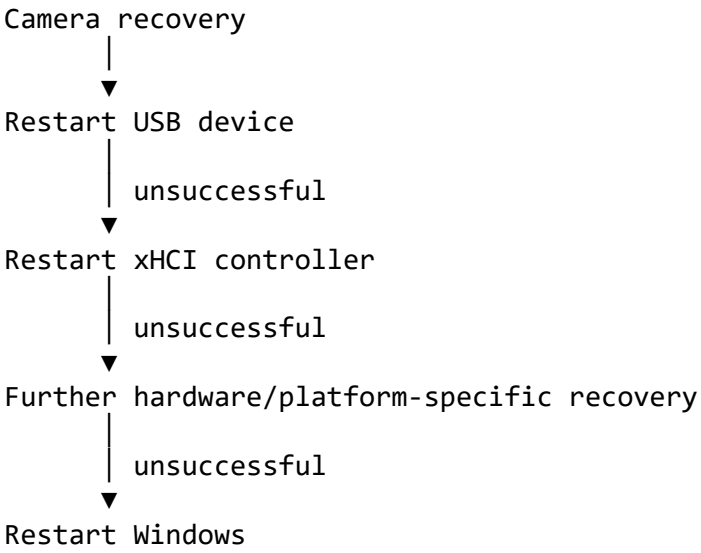
- Camera acquisition stops unexpectedly.
- The camera cannot be reopened by the application.
- Reconnecting the camera does not restore operation.
- Restarting the camera application does not restore operation.
- The device may appear with an error in Windows Device Manager.
- The USB controller may need to be restarted before the camera becomes available again.

The exact behavior depends on the hardware platform, USB controller, Windows version, driver stack, and camera.

3. Recovery Without Rebooting Windows

A complete Windows reboot is usually the most disruptive recovery mechanism.

Before rebooting the entire system, a less intrusive recovery sequence can be attempted:



The appropriate recovery level depends on the actual failure.

For example, if only the camera has stopped responding, restarting the camera device may be sufficient. If the xHCI controller itself is in an error state, restarting the controller may be necessary.

4. Stop the Camera Driver Service Before Restarting the Controller

Practical testing has shown that restarting or disabling the xHCI controller (refer to next chapters) may not be sufficient while the camera driver service is still running.

The **IC4 GenTL U3V service** should therefore be stopped before initiating any xHCI controller recovery operation. This ensures that the camera driver service is no longer actively using the USB subsystem when Windows restarts or disables the controller.

The recommended sequence is:

Stop IC4 GenTL U3V service

↓

Restart or disable xHCI controller

↓

Start IC4 GenTL U3V service

↓

Reinitialize camera acquisition

4.1 Stop the IC4 GenTL U3V Service

Open an elevated PowerShell terminal and execute:

```
Stop-Service ic4-gentl-u3v-service
```

This stops the IC4 GenTL U3V service before the USB host controller is restarted.

4.2 Restart the xHCI Controller

Once the service has been stopped, restart the affected xHCI controller using PnPUtil:

```
pnputil /restart-device "<INSTANCE_ID>"
```

Alternatively, if a disable/enable sequence is required, execute:

```
pnputil /disable-device "<INSTANCE_ID>"
```

Wait briefly, then re-enable the controller:

```
pnputil /enable-device "<INSTANCE_ID>"
```

The controller Instance ID must be obtained from the target system, as described in the following chapter.

4.3 Start the IC4 GenTL U3V Service Again

After the controller has been restarted and Windows has had sufficient time to reinitialize the USB subsystem, start the camera driver service again:

```
Start-Service ic4-gentl-u3v-service
```

The camera application can then enumerate the camera and reinitialize acquisition.

4.4 4.4 Recommended Recovery Sequence

For the tested configuration, the complete recovery sequence is:

```
Stop-Service ic4-gentl-u3v-service

pnputil /restart-device "<INSTANCE_ID>"

Start-Service ic4-gentl-u3v-service
```

If the disable/enable method is used instead:

```
Stop-Service ic4-gentl-u3v-service

pnputil /disable-device "<INSTANCE_ID>"
pnputil /enable-device "<INSTANCE_ID>"

Start-Service ic4-gentl-u3v-service
```

Important:

The service must be stopped before restarting or disabling the xHCI controller. Starting the service again should only be attempted after Windows has completed the controller recovery operation.

The camera application may need to reopen the camera or reinitialize its acquisition stream after the service has been restarted.

5. Using Microsoft's PnPUtil

Windows includes the **PnPUtil.exe** command-line utility for managing Plug and Play devices.

PnPUtil is included with Windows and supports operations including:

- Enumerating devices
- Disabling devices
- Enabling devices
- Restarting devices
- Removing devices
- Scanning for hardware changes

Microsoft provides the `/restart-device` command specifically for restarting a device. The command is available starting with Windows 10 version 2004. Administrator privileges are required for most PnPUtil operations.

6. Identify the xHCI Controller

Before restarting the controller, its Windows **Device Instance ID** must be identified.

Open an elevated Command Prompt and execute:

```
pnputil /enum-devices /connected
```

Alternatively, PowerShell can be used:

```
Get-PnpDevice |
    Where-Object {
        $_.FriendlyName -like "*xHCI*" -or
        $_.FriendlyName -like "*USB 3*"
    } |
    Format-Table Status, Class, FriendlyName, InstanceId
```

A typical controller may appear as:

```
Intel(R) USB 3.20 eXtensible Host Controller - 1.20 (Microsoft)
```

The corresponding Instance ID may look similar to:

```
PCI\VEN_8086&DEV_xxxx&SUBSYS_xxxxxxxx&REV_xx\...
```

The exact ID depends on the hardware platform.

Important:
The actual Instance ID should be obtained from the target system. Do not use a generic or hard-coded ID across different PC platforms.

7. Method 1 – Restart the xHCI Controller

Once the correct Instance ID has been identified, the controller can be restarted using:

```
pnputil /restart-device "<INSTANCE_ID>"
```

For example:

```
pnputil /restart-device "PCI\VEN_8086&DEV_xxxx\..."
```

Microsoft documents `/restart-device` as a supported PnPUtil operation.

After the operation, Windows should reinitialize the device and its associated driver stack.

The camera may subsequently need to be re-enumerated and reopened by the application.

8. Method 2 – Disable and Re-enable the Controller

If a simple restart does not recover the system, a disable/enable sequence can be tested:

```
pnputil /disable-device "<INSTANCE_ID>"
```

Wait briefly and then execute:

```
pnputil /enable-device "<INSTANCE_ID>"
```

For example:

```
pnputil /disable-device "PCI\VEN_8086&DEV_xxxx\..."

timeout /t 2 /nobreak

pnputil /enable-device "PCI\VEN_8086&DEV_xxxx\..."
```

PnPUtil officially supports both `/disable-device` and `/enable-device`. These commands are available starting with Windows 10 version 2004.

This approach is conceptually similar to disabling and subsequently enabling the controller in Windows Device Manager.

9. PowerShell Automation

The recovery process can also be automated using PowerShell.

For example:

```
$controller = Get-PnpDevice |
    Where-Object {
        $_.FriendlyName -like "*xHCI*"
    } |
    Select-Object -First 1

if ($controller) {
    pnputil /restart-device "$($controller.InstanceId)"
}
```

For a production application, however, it is recommended to identify the controller more precisely.

For example, the application could:

1. Enumerate the available xHCI controllers.
2. Identify the controller associated with the camera.
3. Verify the controller's current status.
4. Execute the recovery operation.
5. Wait for Windows to reinitialize the controller.
6. Verify that the camera has returned.
7. Reinitialize the camera acquisition.

10. Native Application Integration

Calling PnPUtil from an application is a practical solution for an initial implementation.

For example, a C++ application could execute:

```
pnputil.exe /restart-device "<INSTANCE_ID>"
```

However, applications that require tighter integration with Windows Plug and Play can use the Windows **SetupAPI** directly.

The relevant mechanism is the DIF_PROPERTYCHANGE device-installation request.

Windows provides the SP_PROPCHANGE_PARAMS structure for this purpose. Microsoft documents DICS_PROPCHANGE as a state change that causes Windows to stop and restart the device. Microsoft also recommends using DICS_PROPCHANGE rather than explicitly specifying DICS_STOP or DICS_START when the objective is to stop and restart a device so that configuration changes take effect.

The general concept is:

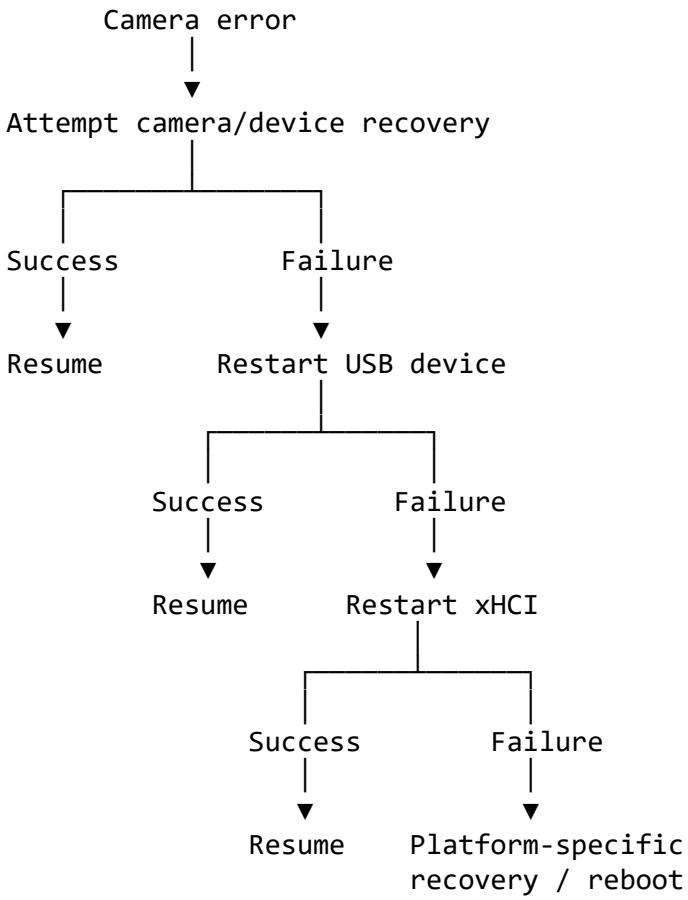
```
Enumerate devices
       |
       ▼
Identify xHCI controller
       |
       ▼
Obtain device information
       |
       ▼
SetupDiCallClassInstaller()
       |
       ▼
DIF_PROPERTYCHANGE
       |
       ▼
DICS_PROPCHANGE
       |
       ▼
Windows stops and restarts device
```

This allows the recovery mechanism to be implemented directly within a native Windows application without relying on an external command-line utility.

11. Recommended Recovery Strategy for Industrial Camera Applications

For an industrial vision application, restarting the entire xHCI controller should preferably not be the first recovery mechanism.

A hierarchical recovery strategy is recommended:



This minimizes the impact on other USB devices connected to the same host controller.

12. Important Considerations

12.1 Administrator privileges

Device-state changes generally require elevated privileges.

An application implementing automatic xHCI recovery should therefore consider how it will obtain the necessary permissions.

Microsoft explicitly states that administrator rights are required for most PnPUtil commands.

12.2 Multiple xHCI controllers

A PC may contain more than one USB host controller.

For example:

```
xHCI Controller #1
 ├── Camera A
 ├── Camera B
 └── USB device

xHCI Controller #2
 ├── Camera C
 └── USB device
```

Restarting the wrong controller can unnecessarily disconnect unrelated USB devices.

The application should therefore identify the controller associated with the affected camera whenever possible.

12.3 Other USB devices may be affected

Restarting an xHCI controller can affect **all USB devices connected through that controller**.

This is particularly important in industrial systems where USB devices such as:

- Cameras
- Encoders
- USB-to-Ethernet adapters
- USB storage
- Input devices
- Measurement equipment

may share the same controller.

The recovery mechanism should therefore be treated as a system-level recovery operation rather than a camera-only reset.

13. Restarting the Controller Is Not Necessarily a Hardware Reset

It is important to distinguish between restarting a Windows device and performing a physical or PCIe-level reset of the hardware.

The PnPUtil and SetupAPI approaches operate through the Windows Plug and Play infrastructure.

Conceptually:

```
Application
     |
     ▼
Windows PnP
     |
     ▼
USB/xHCI driver stack
     |
     ▼
xHCI controller
```

A PnP restart does **not necessarily guarantee that the underlying xHCI hardware is electrically reset**.

If the controller hardware itself has entered a severe or unrecoverable state, restarting the Windows device stack may not be sufficient.

In such cases, recovery may require platform-specific mechanisms, such as a PCIe-level reset or ultimately a system reboot.

14. Recommended Implementation

For most applications, the following approach is recommended as a starting point:

14.1 Step 1 – Detect the camera failure

Detect conditions such as:

- Acquisition timeout
- USB transfer error
- Camera no longer responding
- Camera cannot be reopened

14.2 Step 2 – Attempt camera-level recovery

Attempt to close and reopen the camera or reset the affected USB device.

14.3 Step 3 – Restart the xHCI controller

If the camera cannot be recovered:

```
pnputil /restart-device "<xHCI_INSTANCE_ID>"
```

14.4 Step 4 – Wait for device enumeration

Allow Windows sufficient time to restart the controller and enumerate the connected USB devices.

14.5 Step 5 – Reinitialize the camera

The camera application should then enumerate the camera again and reopen the acquisition stream.

14.6 Step 6 – Escalate if necessary

If the controller remains inaccessible, use a platform-specific recovery mechanism or reboot the system.

15. Example Automated Recovery

A simplified PowerShell implementation could look like this:

```
# Find xHCI controller
$controller = Get-PnpDevice |
    Where-Object {
        $_.FriendlyName -like "*xHCI*"
    } |
    Select-Object -First 1

if (-not $controller) {
    Write-Error "No xHCI controller found."
    exit 1
}

Write-Host "Restarting:"
Write-Host $controller.FriendlyName
Write-Host $controller.InstanceId

# Restart controller
pnputil /restart-device "$($controller.InstanceId)"

if ($LASTEXITCODE -ne 0) {
    Write-Error "Failed to restart xHCI controller."
    exit $LASTEXITCODE
}

Write-Host "xHCI controller restart completed."
```

For a production implementation, additional checks should be added to ensure that the correct controller is selected and that the camera is actually operational after the restart.

16. Conclusion

Windows provides several mechanisms for recovering USB devices without rebooting the entire system.

For modern Windows systems, **PnPUtil is the recommended starting point** because it is included with Windows and provides dedicated commands for restarting, disabling and enabling devices. Microsoft also provides a migration path from the older DevCon utility to PnPUtil.

For a USB 3.x industrial camera application, the recommended recovery sequence is:

```
Camera recovery
       ↓
USB device recovery
       ↓
xHCI controller restart
       ↓
Platform-specific hardware recovery
       ↓
System reboot
```

The `pnputil /restart-device` approach provides a relatively simple way to test whether an xHCI controller can be recovered without restarting Windows.

For applications requiring a fully integrated solution, the Windows SetupAPI can be used to implement device-state changes directly within the application.

Note: The actual effectiveness of xHCI recovery depends on the specific Windows version, USB host-controller hardware, driver implementation, system configuration and nature of the failure. The recovery procedure should therefore be validated on the target hardware before being deployed in an unattended production system.

Revision History

Document Number	Date	Changes
AN20260909-01	15 September 2026	Initial/draft release version of this document.

Headquarters: The Imaging Source Europe GmbH

Überseetor 18, 28217 Bremen, Germany

Phone: +49 421 33591-0

North and South America: The Imaging Source, LLC

4600 Park Road, Suite 470, Charlotte, NC 28209, USA

Phone: +1 704-370-0110

Asia Pacific: The Imaging Source Asia Co. Ltd.

3F., No. 43-7/8, Zhongxing Road,

New Taipei City, Xizhi District 221012, Taiwan

Phone: +886 2-2792-3153

www.theimagingsource.com