Intel® QuickAssist Technology
Software for Windows*

Technical Guide

Revision 1.5

October 2021
You may not use or facilitate the use of this document in connection with any infringement or other legal analysis. You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

All product plans and roadmaps are subject to change without notice.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

© Intel Corporation. Intel, Xeon, Atom, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries.

*Other names and brands may be claimed as the property of others.

Copyright © 2020, Intel Corporation. All rights reserved.
Contents

1 Introduction ...................................................................................................... 7
  1.1 About This Document .............................................................................. 7  
  1.2 Supported Hardware ............................................................................... 7  
  1.3 Supported Operating Systems ................................................................. 7  
  1.4 Supported Software Versions .................................................................. 8  
  1.5 Related Documents and References ....................................................... 9  
  1.6 Terminology ........................................................................................... 9

2 QAT Platform Considerations ............................................................................. 11
  2.1 QAT Device Number .............................................................................. 11  
  2.2 PCI Express Bandwidth .......................................................................... 11  
  2.3 CPU P-States ........................................................................................ 11

3 QAT Driver Installation/Uninstallation ................................................................. 12
  3.1 Unsupported QAT Driver Install/Uninstall Methods .................................... 12  
  3.2 Windows* QAT Package Installation ........................................................ 12  
    3.2.1 Installation Without QAT Hardware ............................................ 12  
    3.2.2 QatSetup Installer ................................................................... 12  
      3.2.2.1 Standalone vs Hyper-V Mode ...................................... 12  
    3.2.3 QAT Install via Command Line.................................................... 13  
      3.2.3.1 QatSetup Install via Command Line ............................ 13  
      3.2.3.2 QAT Driver Install via DevCon .................................. 14  
      3.2.3.3 QAT Driver Install via PnPUtil .................................... 14  
      3.2.3.4 DevCon/PnPUtil Installation Considerations .......... 14
  3.3 QAT Driver Uninstallation ....................................................................... 15  
    3.3.1 QatSetup Uninstall ................................................................... 15  
    3.3.2 QAT Uninstall via Command Line ............................................... 15  
      3.3.2.1 QatSetup Uninstall via Command Line ......................... 16  
      3.3.2.2 QAT Driver Uninstall via DevCon ................................. 16  
      3.3.2.3 QAT Driver Uninstall via PnPUtil .................................. 16  
      3.3.2.4 Devcon/Pnputil Uninstallation Considerations .......... 16
  3.4 QAT Driver Interaction with the Intel® Chipset Drivers ............................... 17  
  3.5 Windows* QAT Files and Services ........................................................... 17  
    3.5.1 Windows* QAT Binaries ............................................................ 17  
    3.5.2 Windows* QAT Services ........................................................... 17  
    3.5.3 Windows* QATzip Files ............................................................. 18  
    3.5.4 Intel® ISA-L ............................................................................ 18  
      3.5.4.1 ISA-L Registry Key .................................................... 18

4 Windows* QAT Diagnostics ............................................................................... 19  
  4.1 Windows* QAT Counters Access – Native and Virtualized ....................... 19  
  4.2 Using Windows* Perfmon Counters for QAT .......................................... 19  
    4.2.1 Rate Counters vs. Interval Counters ........................................ 19  
  4.3 Windows* Perfmon QAT Counter Descriptions ...................................... 20  
    4.3.1 AE Firmware ........................................................................... 20  
    4.3.2 Compression .......................................................................... 20  
      4.3.2.1 Example Compression Counter Usage ....................... 21  
    4.3.3 Cryptography .......................................................................... 22  
      4.3.3.1 ECC ........................................................................ 22
7.6.3.3 OpenSSL*/QAT Engine/Async Nginx* Installation ......... 42
7.6.1.4 Async Nginx* Configuration ........................................... 43
7.6.2 Testing SR-IOV SW Fallback ........................................ 45
7.6.3 Verifying SR-IOV SW Fallback ...................................... 46

8 Support .............................................................................................................. 47

Figures
Figure 1. Installation Options.................................................................................. 13
Figure 2. QatSetup Uninstall .............................................................................. 15

Tables
Table 1. Supported Host Operating Systems ......................................................... 8
Table 2. Supported Guest Operating Systems for SR-IOV ..................................... 8
Table 3. Supported Software Versions ................................................................. 8
Table 4. List of Related Documents ...................................................................... 9
Table 5. Terminology ............................................................................................ 9
Table 6. AE Firmware Counters ......................................................................... 20
Table 7. QAT Hardware Compression and Decompression Counters ................ 20
Table 8. Counters for QAT Hardware ECC Cryptography ................................... 22
Table 9. Counters for QAT Hardware ECDH Cryptography .................................. 22
Table 10. Counters for QAT Hardware ECDSA Cryptography ............................. 23
Table 11. Counters for QAT Hardware RSA Cryptography .................................. 23
Table 12. QATzip Extended Return Codes ......................................................... 30
# Revision History

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Revision Number</th>
<th>Description</th>
<th>Revision Date</th>
</tr>
</thead>
</table>
| 648260          | 1.5.0           | • Updated Table 2. Supported Guest Operating Systems for SR-IOV  
                     • Updated Table 4. List of Related Documents  
                     • Updated Note in Section 3.5.4.1 ISA-L Registry Key  
                     • Updated Section 6.2.1 QATzip Extended Return Codes  
                     • Added Table 12 in Section 6.2.1 QATzip Extended Return Codes  
                     • Updated Code in Section 7.5 QAT Linux* VF Driver Installation  
                     • Updated Code 7.6.1.3 OpenSSL*/QAT Engine/Async Nginx* Installation | October 2021 |
| 648260          | 1.4.0           | • Initial release | August 2021 |
1 Introduction

This document discusses the following topics related to using the Intel® QuickAssist Technology (QAT) Release 1.6.0 Windows* drivers and software. The document is organized in the following sections:

- QAT Driver Installation/Uninstallation
- Windows* QAT Diagnostics
- Windows* QAT Sample Applications
- Example application code using QATzip and CNG for hardware accelerated compression and cryptography
- Brief QATzip overview regarding compression algorithms and data formats

1.1 About This Document

The target audience for this guide are as follows:

- End users interested in utilizing or deploying hardware accelerated compression and cryptography.
- Application developers who want to use Intel® QuickAssist to accelerate compression and cryptographical workloads.

1.2 Supported Hardware

The information in this document pertains to the following devices:

- Intel® QuickAssist Adapter 8960 and 8970
- Intel® Xeon® Scalable First-Generation Platform with Intel® C62x Chipset (with Intel® QAT)
- Intel® Xeon® Scalable Second-Generation Platform with Intel® C62x Chipset (with Intel® QAT)
- Intel® Xeon® Scalable Third-Generation Platform with Intel® C62x Chipset (with Intel® QAT)
- Intel® Xeon® D Platform with Intel® C62x Chipset (with Intel® QAT)
- Intel® Atom® Processor C3000 series (with Intel® QAT)

1.3 Supported Operating Systems

The following table illustrates the supported Host Operating Systems:
## Table 1. Supported Host Operating Systems

<table>
<thead>
<tr>
<th>Host Operating System</th>
<th>Intel® QAT 8960/8970/C62x</th>
<th>Intel® QAT C3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows* Server 2016</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Windows* Server 2019</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Windows* Server 20H2</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Windows* Server 2022</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Windows* 10 Enterprise LTSC</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The following table illustrates the supported Guest Operating Systems for SR-IOV when using the Windows* Physical Function (PF):

## Table 2. Supported Guest Operating Systems for SR-IOV

<table>
<thead>
<tr>
<th>Guest Operating System</th>
<th>Intel® QAT 8960/8970/C62x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows* Server 2016</td>
<td>No</td>
</tr>
<tr>
<td>Windows* Server 2019</td>
<td>Yes</td>
</tr>
<tr>
<td>Windows* Server 2022</td>
<td>No</td>
</tr>
<tr>
<td>Ubuntu* 16.04 LTS, Kernel 4.15</td>
<td>Yes</td>
</tr>
<tr>
<td>Ubuntu* 18.04 LTS, Kernel 4.15</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note:** The only supported architecture for Windows* QAT applications is x86-64.

**Note:** While there may be other Operating System combinations that may work with QAT, they have not been validated by Intel®.

### 1.4 Supported Software Versions

The following table lists the supported software versions that have been validated against the Intel® QAT Windows* Release 1.6 package.

## Table 3. Supported Software Versions

<table>
<thead>
<tr>
<th>Software Name</th>
<th>Package Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Async-Mode Nginx*</td>
<td>0.45</td>
<td>SR-IOV QAT-Engine SW Fallback (Nginx*)</td>
</tr>
<tr>
<td>Intel® ISA-L</td>
<td>2.30</td>
<td>Windows* Compression SW Fallback</td>
</tr>
<tr>
<td>Intel® Linux* QAT Drivers</td>
<td>L4.14-00031</td>
<td>QAT Linux* Driver (for SR-IOV)</td>
</tr>
<tr>
<td>Intel® QAT Engine</td>
<td>0.6.5</td>
<td>SR-IOV QAT-Engine SW Fallback</td>
</tr>
<tr>
<td>Intel® QATzip</td>
<td>V1.0.5</td>
<td>QATzip Compression</td>
</tr>
<tr>
<td>OpenSSL* Github</td>
<td>1.1.1j</td>
<td>SR-IOV QAT-Engine SW Fallback</td>
</tr>
</tbody>
</table>
### 1.5 Related Documents and References

#### Table 4. List of Related Documents

<table>
<thead>
<tr>
<th>Document</th>
<th>Reference/Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Async Mode for Nginx*</td>
<td><a href="https://github.com/intel/async_mode_nginx">https://github.com/intel/async_mode_nginx</a></td>
</tr>
<tr>
<td>Intel® ISA-L Github</td>
<td><a href="https://github.com/intel/isa-i">https://github.com/intel/isa-i</a></td>
</tr>
<tr>
<td>Intel® Linux* QAT Drivers</td>
<td><a href="https://01.org/packet-processing/intel%C2%AE-quickassist-">https://01.org/packet-processing/intel%C2%AE-quickassist-</a></td>
</tr>
<tr>
<td></td>
<td>technology-drivers-and-patches</td>
</tr>
<tr>
<td>Intel® QAT Engine</td>
<td><a href="https://github.com/intel/QAT_Engine">https://github.com/intel/QAT_Engine</a></td>
</tr>
<tr>
<td>Intel® QATzip Github</td>
<td><a href="https://github.com/intel/QATzip">https://github.com/intel/QATzip</a></td>
</tr>
<tr>
<td>Microsoft* DevCon Github</td>
<td><a href="https://github.com/microsoft/Windows-driver-samples/tree/master/setup/devcon">https://github.com/microsoft/Windows-driver-samples/tree/master/setup/devcon</a></td>
</tr>
<tr>
<td>Microsoft* Powershell Github</td>
<td><a href="https://github.com/powershell/powershell">https://github.com/powershell/powershell</a></td>
</tr>
<tr>
<td>OpenSSL* Github</td>
<td><a href="https://github.com/openssl/openssl">https://github.com/openssl/openssl</a></td>
</tr>
</tbody>
</table>

### 1.6 Terminology

#### Table 5. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>Also known as Request Size, this is the data size being sent to the QAT</td>
</tr>
<tr>
<td></td>
<td>hardware.</td>
</tr>
<tr>
<td>Calgary corpus</td>
<td>A collection of text and binary data files commonly used for comparing</td>
</tr>
<tr>
<td></td>
<td>data compression algorithms</td>
</tr>
<tr>
<td>Chunk Size</td>
<td>Also known as hardware Buffer Size (QzSessionParams_T.hw_buff_sz). The data</td>
</tr>
<tr>
<td></td>
<td>size being processed by the QAT hardware.</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>A lightweight command used in the Powershell environment.</td>
</tr>
<tr>
<td>Deflate</td>
<td>Lossless data compression file format that uses LZSS and Huffman coding</td>
</tr>
<tr>
<td></td>
<td>specified in RFC 1951.</td>
</tr>
<tr>
<td>Forks</td>
<td>Also corresponds to Max Forks (QzSessionParams_T.max_forks). This is the</td>
</tr>
<tr>
<td></td>
<td>number of requests that can happen concurrently.</td>
</tr>
<tr>
<td>ISA-L</td>
<td>Intel® Intelligent Storage Acceleration Library</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PF</td>
<td>Physical Function</td>
</tr>
<tr>
<td>PKE</td>
<td>Public Key Encryption</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Powershell</td>
<td>Cross-platform command-line shell and scripting language using the .NET Common Runtime. Everyone should be using it.</td>
</tr>
<tr>
<td>QAT</td>
<td>Intel® QuickAssist Technology</td>
</tr>
<tr>
<td>SR-IOV</td>
<td>Single Root I/O Virtualization</td>
</tr>
<tr>
<td>Threshold Size</td>
<td>Also known as Input Size Threshold (QzSessionParams_T.input_sz_threshold). Block sizes smaller than this will be routed to software.</td>
</tr>
<tr>
<td>VF</td>
<td>Virtual Function</td>
</tr>
<tr>
<td>VHD(x)</td>
<td>Virtual Hard Disk, VHDx is the successor file format.</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WDK</td>
<td>Windows* Driver Kit</td>
</tr>
<tr>
<td>WPP</td>
<td>Windows* Software Trace Pre-processor</td>
</tr>
</tbody>
</table>
2 QAT Platform Considerations

This section describes platform configuration options that may affect QAT hardware performance.

2.1 QAT Device Number

The number of QAT devices may vary depending on the platform used. As such, having more QAT devices generally increases performance linearly, pending other system bottlenecks.

2.2 PCI Express Bandwidth

For optimal performance, the QAT device should have 8x PCI Express 3.0 lanes. For example, a system with three QAT devices should have 24x PCI Express 3.0 lanes. Using anything less may result in performance degradation in I/O bound operations, such as compression and decompression.

2.3 CPU P-States

Enabling CPU P-States may lead to a noticeable drop in (de)compression performance. However, QAT hardware accelerated performance should still be higher than using software.

For optimal QAT device performance, the recommendation is to disable P-States.

This has been observed but may not limited to the Intel® Xeon® Scalable First and Second generation platforms.
3 QAT Driver Installation/Uninstallation

The QAT Windows* driver package can be installed in several different ways. Intel® recommends using the Setup MSI installer as it is the complete way to install and bring up the QAT driver. Intel® also recommends using the Setup MSI installer to fully remove the QAT driver and its components. As with all Windows* driver installations, administrator privileges are required.

**Note:** The Windows* QAT driver is meant to be installed as a complete package. Mix matching of any components within this package is not supported and may lead to undesirable behavior.

3.1 Unsupported QAT Driver Install/Uninstall Methods

Per Microsoft* guidelines, Intel® does not recommend using Win32_Product to install or uninstall via PowerShell. Doing so maybe lead to undesirable behavior.


3.2 Windows* QAT Package Installation

All the examples in this section assumes that the user has extracted the QAT driver package into the following folder: `C:\temp\QatDriver`

3.2.1 Installation Without QAT Hardware

In cases where there is no Intel® QAT hardware in the system, the Windows* QAT package will attempt to install qatzip.dll into the system directory.

3.2.2 QatSetup Installer

Requires MS VC Redistributable. If the redistributable is not available, it will automatically be installed.

This is the recommended installation method for Windows* Operating Systems with a Desktop Environment. The QatSetup.exe installer is located in the path: `C:\temp\QatDriver\QuickAssist\Setup\QatSetup.exe`

3.2.2.1 Standalone vs Hyper-V Mode

Starting with QAT Windows* Release 1.5, setup will prompt for two different installation modes: Standalone or Hyper-V.
3.2.2.1.1 **Standalone Mode**

This will install all the QAT Windows* drivers and services that allows for Intel® hardware offload for compression and crypto workloads. It is still possible to run SR-IOV workloads should the device be put in SR-IOV mode.

This installation mode will also attempt to install ISA-L (required for compression software fallback). This requires the nuget application to be installed.

3.2.2.1.2 **Hyper-V Mode**

Only the QAT Windows* base driver will be installed. In addition, the installer will put all the Intel® QAT devices in SR-IOV mode. It is not possible to use Intel® QAT hardware offloading on the Host OS.

*Note:* Hyper-V Mode installation requires a system restart.

3.2.3 **QAT Install via Command Line**

For environments without the Windows* Desktop Environment, there are several options to install the driver via the command line.

*Note:* When installing the QAT driver using DevCon or PnPUtil, it is up to the end user to correctly install and initialize the requisite files and services or it may lead to undesirable behavior.

3.2.3.1 **QatSetup Install via Command Line**

The QatSetup.exe mentioned in 3.2.1 has command line capability. To get a list of all available options:

```
QatSetup.exe --help
```
An example silent, no touch installation using QatSetup.exe would be:
QatSetup.exe /passive /qn

3.2.3.2 QAT Driver Install via DevCon

DevCon is a Windows* command line application that can display detailed device information as well as installing, configuring, and removing devices.

It is available in binary format with the Microsoft* WDK or available as source code:
https://github.com/microsoft/Windows-driver-samples/tree/master/setup/devcon

Example to install a driver via DevCon:
devcon.exe upgrade <InfFile> <DeviceId>

Where InfFile is the full path to the Inf file of the driver to be installed and DeviceId is the DeviceId of the associated device to be installed. Example:
devcon.exe upgrade C:\temp\QatDriver\QuickAssist\Driver\x64\icp_qat.inf
PCI\VEN_8086&DEV_37C8

Note: Using devcon ‘install’ option instead of ‘upgrade’ may result in a ghost device. In this situation, the user should remove the ghost device.

3.2.3.3 QAT Driver Install via PnPUtil

The application PnPUtil is a command line based tool that allows users to perform actions on driver packages. This is included in every version of Windows* since Vista.

Example to install the QAT driver via PnPUtil:
pnputil.exe /add-driver <InfFile> /install

Where InfFile is the full path to the Inf file of the driver to be installed. Example:
pnputil.exe /add-driver
C:\temp\QatDriver\QuickAssist\Driver\x64\icp_qat.inf /install

3.2.3.4 DevCon/PnPUtil Installation Considerations

Using DevCon and PnPUtil to install the QAT driver will necessitate additional steps before the device can be used:

1. Make the ICP FW folder in the System32 drivers path:
   New-Item C:\Windows\System32\drivers\icpQATFW -ItemType Directory

2. Copy the QAT firmware files to the System32 drivers path:
   Copy-Item -Path "C:\temp\QatDriver\*" -Destination "C:\Windows\System32\drivers\icpQATFW"

3. Create/start the compression service (if compressed is to be used):
   sc.exe create cfqat type= kernel start= boot binpath= C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression\CfQat.sys
   sc.exe start cfqat

4. Create/start the kernel mode crypto service (if cryptographical services is to be used):
   sc.exe create cpmprov type= kernel start= boot binpath= C:\Windows\System32\drivers\CPMProv.sys
5. Create/start the user mode crypto service (if cryptographical services is to be used):
   `sc.exe create cpmprovuser type= kernel start= boot binpath= C:\Windows\System32\drivers\CPMProvUser.sys`
   `sc.exe start cpmprovuser`

6. Make sure all the requisite services can start successfully.
7. Start the QAT device driver. The device status should be 'OK'.

3.3 **QAT Driver Uninstallation**

All the examples in this section assumes that the user has extracted the QAT driver package into the following folder: `C:\temp\QatDriver`

3.3.1 **QatSetup Uninstall**

From the Windows* Desktop, the QAT driver can be uninstalled from the Control Panel via Programs Uninstall. Select the entry with Name "Intel® QuickAssist Technology".

![Figure 2. QatSetup Uninstall](image)

3.3.2 **QAT Uninstall via Command Line**

For environments without the Windows* Desktop Environment, there are several options to uninstall the driver via the command line.

**Note:** When uninstalling the QAT driver package using DevCon or PnPUtil, it is up to the end user to correctly remove all the requisite files and services or it may lead to undesirable behavior.
3.3.2.1 QatSetup Uninstall via Command Line

The QatSetup.exe mentioned in 3.2.1 has command line capability. To get a list of all available options:
QatSetup.exe --help

An example silent, no touch uninstall using QatSetup.exe would be:
QatSetup.exe /x /qn

3.3.2.2 QAT Driver Uninstall via DevCon

DevCon is a Windows* command line application, for more information, see Section 3.2.2.2.

Example to uninstall a driver via DevCon:
devcon.exe remove <DeviceId>

Where DeviceId is the DeviceId of the associated device to be uninstalled. Example:
devcon.exe remove PCI\VEN_8086&DEV_37C8

3.3.2.3 QAT Driver Uninstall via PnPUtil

The application PnPUtil is a command line-based tool that allows users to perform actions on driver packages. This is included in every version of Windows* since Vista.

PnPUtil uninstall functionality requires Windows* version 1607 or later.

Example to uninstall the QAT driver via PnPUtil:
pnputil.exe /delete-driver <oem#.inf> /uninstall /force /reboot

Where the oem#.inf is the inf file generated when installing the QAT driver using the DevCon or PnPUtil. Example:
pnputil.exe /delete-driver oem7.inf /uninstall /force /reboot

3.3.2.4 Devcon/PnPUtil Uninstallation Considerations

Using DevCon and PnPUtil to uninstall the QAT driver will necessitate additional steps if all the files and services are removed:

1. Remove the QAT firmware files:
   Remove-Item -Recurse C:\Windows\System32\drivers\icpQATFW

2. Remove the compression service (if installed):
   sc.exe stop cfqat
   sc.exe delete cfqat

3. Remove the kernel mode crypto service (if installed):
   sc.exe stop cpmprov
   sc.exe delete cpmprov

4. Remove the user mode crypto service (if installed):
   sc.exe stop cpmprovuser
   sc.exe delete cpmprovuser

5. (Optional) Remove the oem.inf driver entry from the driver store. If this driver entry from the driver store is not removed, then Windows* may automatically install the QAT device driver without the necessary prerequisite services and firmware files.
6. A reboot of the system may be necessary.

3.4 **QAT Driver Interaction with the Intel® Chipset Drivers**

The Intel® Chipset Drivers contains a null driver for the Intel® QAT device. While this QAT null driver may successfully initialize, the only functionality is cosmetic.

The effects of installing the Intel® Chipset drivers with the Intel® QAT drivers are as follows:

1. In the registry entry for the QAT device, there may be double the UUID entries.
2. In Powershell or using the Win32_PnPEntity class, there may be double the QAT devices reported. The null devices should have a non-OK Status.
3. When uninstalling the Intel® QAT driver package, the QAT device may revert to the Intel® Chipset QAT null driver.

3.5 **Windows* QAT Files and Services**

3.5.1 **Windows* QAT Binaries**

The following are QAT Windows* system files that are installed using the default Windows* QAT Setup installer options.

Located in ‘C:\Program Files\Intel\Intel® QuickAssist Technology\Compression’:

- CfQat – Windows* QAT binary used for QAT hardware accelerated compression and decompression.

Located in ‘Drivers’, where the parent directory is defined by ‘[System.Environment]::SystemDirectory’:

- CPMProv – Windows* QAT binary for QAT hardware accelerated cryptographical offloads in kernel mode.
- CPMProvUser – Windows* QAT binary for QAT hardware accelerated cryptographical offloads in user mode.

3.5.2 **Windows* QAT Services**

The following are QAT Windows* services files that are installed and running using the default Windows* QAT Setup installer options. See Section 3.5.1 for a description of what each service does.

- CfQat
- CPMProv
- CPMProvUser

It is recommended to use the Service Control application to manage these services.
Note: Intel® recommends disabling or removing services that will not be in use.

3.5.3 Windows® QATzip Files

The following are Windows® QATzip files that are installed using the default Windows® QAT Setup installer options. They are located in -

'C:\Program Files\Intel\Intel® QuickAssist Technology\Compression\Library' :

- qatzip.dll – The Windows® QATzip Dynamic Link Library.
- qatzip.lib – The Windows® QATzip DLL library file to hook into the DLL.

These files may be necessary when building custom QATzip based applications.

3.5.4 Intel® ISA-L

Starting with QAT Windows® Release 1.6, the QAT installer package will attempt to install the ISA-L DLL. This requires the system to have nuget package manager installed. If nuget is not present on the system, ISA-L will not be installed.

3.5.4.1 ISA-L Registry Key

Windows® QATzip will attempt to use ISA-L specified at the location defined by this registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Intel\QAT\SWFallback\Location

The Location value can be modified to change the location of ISA-L.

Note: If the ISA-L Location registry key is not present, QATzip will attempt to look for ISA-L in the system directory.
4 Windows* QAT Diagnostics

4.1 Windows* QAT Counters Access – Native and Virtualized

The counters below are supported in Windows* Perfmon.

The AE Firmware counters are available in Windows* native OS operation and in the Windows* Host OS partition when running Hyper-V and assigning Virtual Functions (VFs) to a guest VM.

All other Windows* counters (excluding the above firmware counters) are available only in the OS partition that is running the QAT requests. Therefore, these counters are available in the Host partition if the QAT device is running requests in the host partition. If the QAT device is assigned to a guest or guests, then the counters are available in the Windows* Guest VM.

Note: There may be unimplemented counters in the manifest file.

4.2 Using Windows* Perfmon Counters for QAT

Windows* Counters are implemented using the Windows* Management Interface (WMI). The counters are visible using the Windows* Perfmon GUI and are accessible using all the standard Windows* Logging Features.

Counters provide the ability to determine the amount of data being processed by QAT as well as aid in determining what application changes may be needed to more effectively utilized QAT.

4.2.1 Rate Counters vs. Interval Counters

There are two main types of counter used in Windows* for QAT. They are interval counters and rate counters. The interval counters report the number of events that happened in the Windows* Perfmon Logging Interval. Rate counters report the average number of events per second for the logging interval.

When using a logging interval of 1 second, these two types of counters will report the same information when they are counting the same metric. For instance, the "Compression Bytes In per Interval" counter and the "Compression Throughput" Counter will report the same number when the Perfmon logging interval is 1 second. As an example, if 1 MiB of data is compressed in the 1 second interval, the throughput will be reported as 1,000,000 in the Compression Throughput counter and is interpreted as 1,000,000 bytes per second for that interval. The Compression Bytes In per Interval counter will also report 1,000,000 and is interpreted as 1,000,000 bytes in that reporting interval.

For the same example above, if the reporting interval is 5 seconds for Windows* Perfmon, and the QAT work continues to be 1 MiB / second, the Compression Throughput counter will report 1,000,000 bytes per second because there was 5MiB
compressed divided by 5 seconds for a Rate 1,000,000 bytes per second. The Compression Bytes In per Interval counter will read 5,000,000 bytes because there was 5,000,000 bytes processed in the 5 second interval.

Both counter types are provided as a convenience to those who want to post process data providing more flexibility and ease of use.

The queue depth counter, and total wait time counters are interval counters.

### 4.3 Windows* Perfmon QAT Counter Descriptions

Available counters within Windows* Perfmon are detailed below.

#### 4.3.1 AE Firmware

These counters are available in Windows* native OS operation and in the host OS when running virtualized on Hyper-V regardless of what the guest OS may be.

**Table 6. AE Firmware Counters**

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Retries</td>
<td>Request retries per second. A retry indicates that the driver should try to submit the request again because the previous attempt to submit was not successful.</td>
</tr>
<tr>
<td>AE Firmware Responses</td>
<td>Responses per second from QAT for all responses for the ring bundle</td>
</tr>
<tr>
<td>AE Firmware Requests</td>
<td>Requests per second to QAT for all requests for the ring bundle</td>
</tr>
</tbody>
</table>

#### 4.3.2 Compression

The following counters are available for QAT hardware compression and decompression.

**Table 7. QAT Hardware Compression and Decompression Counters**

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Decomp Wait Time (ms)</td>
<td>This is the sum of the wait time for all decompression requests on a queue for the interval of collection. The average latency of decompression requests for each interval can be calculated by dividing this counter by the “Decompression Completed Ops per Interval” counter to get average latency (ms) per decompression request</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Total Comp Wait Time (ms)</td>
<td>This is the sum of the wait time for all compression requests on a queue for the interval of collection. The average latency of compression requests for each interval can be calculated by dividing this number by the “Compression Completed Ops per Interval” counter to get average latency (ms) per compression request.</td>
</tr>
<tr>
<td>Decompression Completed Ops per Interval</td>
<td>A decompression completion is the number of requests (chunk size) sent to the QAT hardware and completed within the specified interval time.</td>
</tr>
<tr>
<td>Decompression Bytes Out per Interval</td>
<td>Number of decompressed output bytes from QAT as result of decompression operations within the specified interval time.</td>
</tr>
<tr>
<td>Compression Completed Ops per Interval</td>
<td>A compression completion is the number of requests (chunk size) sent to the QAT hardware and completed within the specified interval time.</td>
</tr>
<tr>
<td>Compression Byte In per Interval</td>
<td>Number of input bytes sent to the QAT hardware to be compressed within the specified time interval used in logging.</td>
</tr>
<tr>
<td>Comp and Decomp Average Queue Depth</td>
<td>Reports the average queue depth over the collection interval of all comp and decomp requests in the queue.</td>
</tr>
<tr>
<td>Decompression Errors</td>
<td>The decompression errors returned to the QAT driver.</td>
</tr>
<tr>
<td>Decompression Completions/sec</td>
<td>A decompression completion is the number of requests (chunk size) sent to the QAT hardware and completed.</td>
</tr>
<tr>
<td>Decompression Throughput</td>
<td>The uncompressed bytes out per second resulting from decompression operations.</td>
</tr>
<tr>
<td>Recovered CNVR Ops per Interval</td>
<td>This is not supported for this release.</td>
</tr>
<tr>
<td>Compression Errors</td>
<td>The compression errors returned to the QAT driver.</td>
</tr>
<tr>
<td>Compression Completions/Sec</td>
<td>A compression completion is the number of requests (chunk size) sent to the QAT hardware and completed.</td>
</tr>
<tr>
<td>Compression Throughput</td>
<td>The compression bytes per second (bytes / second) measured by the uncompressed bytes</td>
</tr>
<tr>
<td>Total Comp and Decomp Requests Queued per Second</td>
<td>This is the average requests queued per second (compression and decompression combined) (queued ops/second)</td>
</tr>
</tbody>
</table>

### 4.3.2.1 Example Compression Counter Usage

Using ‘compression bytes in per interval’ counter while compressing Calgary corpus:

```powershell
Get-Counter -Counter "\Intel QuickAssist Accelerator Compression(*)\Compression Bytes In per Interval" -SampleInterval 5 -MaxSamples 1
```
Note that the resultant value is the exact length of the Calgary corpus.

4.3.3 Cryptography

The following counters are available for QAT hardware cryptography.

4.3.3.1 ECC

The following counters are available for QAT hardware ECC cryptography.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Intel QuickAssist Accelerator ECC(*)\ECC Point Verify Errors/sec</td>
<td>ECC Point verify errors returned to QAT driver.</td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator ECC(*)\ECC Point Multiply Errors/sec</td>
<td>ECC Point Multiply errors returned to QAT driver.</td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator ECC(*)\ECC Point Verify Operations/sec</td>
<td>ECC Point Verify successful operations per second.</td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator ECC(*)\ECC Point Multiply Operations/sec</td>
<td>ECC Point Multiple successful operations per second.</td>
</tr>
</tbody>
</table>

4.3.3.2 ECDH

The following counters are available for QAT hardware ECDH cryptography.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Intel QuickAssist Accelerator ECDH(*)\ECDH Point Multiply Errors/sec</td>
<td>ECDH Point Multiply errors returned to QAT driver.</td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator ECDH(*)\ECDH Point Multiply Operations/sec</td>
<td>ECDH Point Multiple successful operations per second.</td>
</tr>
</tbody>
</table>

4.3.3.3 ECDSA

The following counters are available for QAT hardware ECDSA cryptography.
Table 10. Counters for QAT Hardware ECDSA Cryptography

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Verify errors returned to QAT driver.</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Verify Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Signature RS errors returned to QAT driver.</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Sign RS Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Signature S errors returned to QAT driver.</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Sign S Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Signature R errors returned to QAT driver.</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Sign R Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Verify successful operations per second.</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Verify Operations/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Signature RS successful operations per</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Sign RS</td>
<td>second.</td>
</tr>
<tr>
<td>Operations/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Signature S successful operations per</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Sign S Operations/sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>second.</td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>ECDSA Signature R successful operations per</td>
</tr>
<tr>
<td>ECDSA(*)\ECDSA Sign R Operations/sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>second.</td>
</tr>
</tbody>
</table>

4.3.3.4 RSA

The following counters are available for QAT hardware RSA cryptography.

Table 11. Counters for QAT Hardware RSA Cryptography

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>RSA Key Generation errors returned to QAT driver.</td>
</tr>
<tr>
<td>RSA(*)\RSA Key Generation Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>RSA Encrypt errors returned to QAT driver.</td>
</tr>
<tr>
<td>RSA(*)\RSA Encrypt Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>RSA Decrypt errors returned to QAT driver.</td>
</tr>
<tr>
<td>RSA(*)\RSA Decrypt Errors/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>RSA Key Generation successful operations per</td>
</tr>
<tr>
<td>RSA(*)\RSA Key Generation</td>
<td>second.</td>
</tr>
<tr>
<td>Operations/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>RSA Encrypt successful operations per second.</td>
</tr>
<tr>
<td>RSA(*)\RSA Encrypt Operations/sec</td>
<td></td>
</tr>
<tr>
<td>\Intel QuickAssist Accelerator</td>
<td>RSA Decrypt successful operations per second.</td>
</tr>
<tr>
<td>RSA(*)\RSA Decrypt Operations/sec</td>
<td></td>
</tr>
</tbody>
</table>
5 Windows* QAT Sample Applications

The installation of the Windows* QAT driver includes two sample applications and a QATzip static and dynamic library. The purpose of these sample applications is to demonstrate the capabilities of QAT hardware accelerated compression and cryptography.

5.1 Parcomp

Parcomp is the compression sample application that uses QATzip. The purpose of this sample application is to demonstrate the capabilities of QATzip compression and decompression. By default, it is located in the following folder:

C:\Program Files\Intel\Intel® QuickAssist Technology\Compression

For a list of all the parameterizations of parcomp:
parcomp.exe –help

The parcomp sample application is limited to output files of approximately 1.3 GiB, which is the current Windows* QATzip implementation limitation.

5.1.1 Parcomp General Examples

The following are general examples for using the parcomp sample application.

Note: For provider 'qat', it is up to the user to store the appropriate metadata for decompression, otherwise decompressing the data may result in bad data.

5.1.1.1 Parcomp Compressing, Decompressing, and SHA256 Check Using DEFLATE_RAW

Compressing 'calgary' file into 'calgary_compressed' with DEFLATE_RAW, Level 1, 64KB Chunk Size:
parcomp.exe -i C:\temp\calgary -o C:\temp\calgary_compressed -p qat -l 1 -c 64
Decompressing ‘calgary_compressed’ to ‘calgary_decompressed’ (note that compression level is not required) and using Powershell to verify SHA256 hash:
```
parcomp.exe -i C:\temp\calgary_compressed -o C:\temp\calgary_decompressed -p qat -c 64 -d

(Get-FileHash "C:\temp\calgary_decompressed").Hash -eq (Get-FileHash "C:\temp\calgary").Hash
```

5.1.1.2 Parcomp Compressing and Decompressing Using DEFLATE_GZIPEXT

Compressing ‘calgary’ file into ‘calgary_compressed’ with DEFLATE_GZIPEXT Level 1, 64KB Chunk Size and decompressing ‘calgary_compressed’ to ‘calgary_decompressed’ (note that compression level or chunk size is not required since the relevant information is in the gzip headers):
```
parcomp.exe -i C:\temp\calgary -o C:\temp\calgary_compressed -p qatzipext -l 1 -c 64
parcomp.exe -i C:\temp\calgary_compressed -o C:\temp\calgary_decompressed -p qatgzipext -d
```
5.1.1.3 Parcomp Compressing and Decompressing with Multiple Independent Threads

For optimal performance, the parcomp sample application cause use multiple independent compression and decompression threads.

This example uses Silesia to demonstrate > 70 Gbps compression performance and > 120 Gbps decompression performance:

```
parcomp.exe -i C:\temp\silesia -o c:\temp\silesia_compressed -p qat -1 1 -c 64 -k 4096 -Q -t 9 -j 60 -n 100
parcomp.exe -i C:\temp\silesia_compressed0 -o c:\temp\silesia_decompressed -p qat -d -c 64 -k 4096 -Q -t 9 -j 60 -n 100
```

*Note:* Actual performance will vary depending on the target platform.
5.1.2 Parcomp SW Fallback Using ISA-L

The currently supported algorithms for software fallback are DEFLATE_4B, DEFLATE_GZIP, and DEFLATE_GZIP_EXT. The primary use cases for SW Fallback is a threshold scenario, in the event of a QAT hardware failure, or during QAT driver servicing/upgrades.

If the criteria for SW Fallback is met, the request will be rerouted to software using the ISA-L igzip library.

5.1.2.1 Parcomp SW Threshold Example

Using compression threshold, all requests are routed to software if the block size (request size) is smaller than a certain user input value.

Compressing ‘calgary’ file into ‘calgary_compressed’ with DEFLATE_RAW, Level 1, 64KB Chunk Size/512KB Block Size with SW threshold of 256KB:

```
parcomp.exe -i C:\temp\calgary -o C:\temp\calgary_compressed -p qat -l 1 -c 64 -k 512 -FT 256
```

The Calgary corpus is 3,251,493-byte length. Since the block size is 512KB, that means the QAT hardware will receive 512KB of data per request and then break it down into 64KB chunks for processing. The last request size will be 105,765-bytes (3,251,493 % (512 * 1,024)). Since that is under the threshold size of 256KB, the last request will be routed to igzip software.

Using the Get-Counter command during compression:

```
Get-Counter -Counter "\Intel QuickAssist Accelerator Compression(*)\Compression Bytes In per Interval" -SampleInterval 5 -MaxSamples 1
```
Note that the Compression Bytes observed is exactly 105,765-Bytes less than the Calgary corpus, or the exact size of the last request size.

### 5.1.2.2 Parcomp SW Fallback Example

Using SW Fallback in a non-threshold scenario, the requests will be rerouted to igzip software only if a hardware error occurs.

**Note:** Intel® does not recommend intentionally creating hardware errors.

Compressing 'calgary' file into 'calgary_compressed' with DEFLATE_RAW, Level 1, 64KB Chunk Size with SW fallback:

```
parcomp.exe -i C:\temp\calgary -o C:\temp\calgary_compressed -p qat -l 1 -c 64 -FB
```

### 5.2 Cngtest

Cngtest is the cryptographical sample application that can provide QAT hardware accelerated cryptography. The sample application uses the CNG framework. Cngtest does cryptography in memory using generated keys (where applicable).

By default, Cngtest is in the following folder:

```
C:\Program Files\Intel\Intel® QuickAssist Technology\Crypto\Samples\bin
```

For a list of all the parameterizations of Cngtest:

```
cngtest.exe --help
```

#### 5.2.1 CNG Scaling

The Microsoft* CNG framework is synchronous in nature. To obtain optimal performance, it is recommended to use multiple independent threads (numThreads parameter in Cngtest) to parallelize the workload. The optimal number of threads is dependent on the number of Intel® QAT devices available in the system.

**Note:** When creating more than 100 threads per Intel® QAT device, there may be a reduction in observed performance.

#### 5.2.2 Cngtest General Examples

Running RSA, key size 2048, oaep padding, in kernel mode encrypt only:

```
cngtest.exe -provider=qa -algo=rsa -keylength=2048 -padding=oaep -encrypt
```
Running ECDH curve 521, in user mode, all operations:

cngtest.exe -provider=qa -algo=ecdh_521 -user

Running ECDSA curve 521, verify operations:

cngtest.exe -provider=qa -algo=ecdsa_521 -verify

### 5.2.3 Cngtest Fallback

In the event of a hardware failure, cngtest will re-route the request to software using the CNG framework.

*Note:* Intel® does not recommend intentionally creating hardware errors.
6 QATzip API Guide

6.1 Limitations

The QATzip API is limited to output files of approximately 1.3 GiB.

There are some functions in the QATzip API for Windows* that have not been implemented:
- qzMemFindAddr
- qzCompressCrc
- qzCompressCrcExt
- qzCompressStream
- qzDecompressStream
- qzEndStream

6.2 Using QATzip Without QAT

Starting with QAT Windows* Release 1.5, it is possible to use the QATzip API without the underlying Intel® QAT hardware. This provides end-users with the flexibility to deploy a single software solution that can be used on servers without Intel® QAT for compatibility or it can be used on servers with Intel® QAT for performance.

The requirement is to set the parameter sw_backup to 1 and have the ISA-L DLL in the Windows* System32 directory or location defined by the registry key in 3.5.4.

6.2.1 QATzip Extended Return Codes

Starting with QAT Windows* Release 1.6, new APIs were added (qzCompressExt and qzDecompressExt) that includes extended return information. The ext_rc is can be read as follows.

Table 12. QATzip Extended Return Codes

<table>
<thead>
<tr>
<th>Bits</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:5</td>
<td>Bits</td>
<td></td>
<td>Undefined, reserved for future use.</td>
</tr>
<tr>
<td>4</td>
<td>Bit</td>
<td>QZ_SW_EXECUTION_BIT</td>
<td>Value 1: The operation was executed using the software provider. Value 0: the operation was executed using QAT hardware.</td>
</tr>
<tr>
<td>3</td>
<td>Bit</td>
<td></td>
<td>Undefined, reserved for future use.</td>
</tr>
<tr>
<td>2</td>
<td>Bit</td>
<td></td>
<td>Undefined, reserved for future use.</td>
</tr>
<tr>
<td>1</td>
<td>Bit</td>
<td></td>
<td>Undefined, reserved for future use.</td>
</tr>
<tr>
<td>Bits</td>
<td>Type</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>0</td>
<td>Bit</td>
<td></td>
<td>Undefined, reserved for future use.</td>
</tr>
</tbody>
</table>

**Note:** If qzCompressExt/qzDecompressExt is given a null ext_rc, the ext_rc will be null. This would be the same as calling the legacy qzCompress/qzDecompress.

**Note:** Code to determine the value of the ext_rc defined bits must not be written with assumptions about the values of the currently undefined fields in the parameter. For example, to get the value of QZ_SW_EXECUTION_BIT, it is recommended to use the QZ_SW_EXECUTION macro defined in qatzip.h.

### 6.3 QzSessionParams_T Overview

This section contains a brief overview of the member properties of the QzSessionParams_T struct in qatzip.h.

- `huffman_hdr` – Dynamic or static Huffman headers.
- `direction` – Direction of data (e.g., compression/decompression/both).
- `data_fmt` – The data format for the selection compression algorithm. See Section 6.4 regarding the Deflate data formats.
- `comp_lvl` – The compression level.
- `comp_algorithm` – The compression algorithm.
- `max_forks` – The QAT Windows* implementation that determines the maximum number of concurrent requests to hardware per QATzip compress or decompress operation. The default number is 30. Performance gains can be seen in some cases up to 60 though the gains generally decrease as the number increased from 30 to 60. Anything greater than 60 generally results in very little change in performance. This is also known as MaxOutstandingJobs in the parcomp sample application.

**Note:** The number of total jobs submitted to QAT hardware per QATzip compress operation is equal to the input buffer size / hw_buff_size rounded up to the next larger integer. The result of this is that if the typical input buffer size is ‘small’, it may not be necessary to have 30 jobs. For instance, if the max offload size for an application is 256KB, and the chunksize is 64KB, then the max_forks value only needs to be 4 to attain maximum performance.

- `sw_backup` – Enable or disable software fallback. The input_sz_thrshold will override this (e.g., in the event the request size is smaller than the input_sz_thrshold, the request will still go to software even when sw_backup is 0).
- `hw_buff_sz` – The size of data (job) being sent to QAT hardware. This is also known as chunk size in the parcomp sample application. It has been observed that for most data sets, 64KB hw_buff_sz is optimal (and is the default value).

**Note:** It is up to the user to determine what the optimal compression settings are, as workloads vary.

- `input_sz_thrshold` – The threshold size of the request size in which software is to be used. This is also known as Fallback Threshold in the parcomp sample application. The maximum size is $2^{31}$ Bytes. This does not work if sw_backup is not enabled.
The default value is 1 KiB. If the hw_buff_sz is smaller than this value, QAT hardware will never be used.

**Note:** To completely disable software fallback, set the input_sz_thrshold to 0.

**Note:** It is up to the user to determine what the optimal threshold settings are, as workloads vary. It is also a function of acceptable CPU utilization, latency requirements, as well as throughput.

### 6.4 Compression Algorithms

QATzip for Windows* supports the Deflate compression algorithm.

QATzip supports compression levels 1 through 9. However, since Deflate only supports levels 1 through 4, levels 5 through 9 are internally remapped to level 4.

#### 6.4.1 QZ_DEFLATE

This compression algorithm will compress incoming requests into Deflate standard blocks. For the various formats that can be used with QZ_DEFLATE, see Section 6.4.

#### 6.4.2 QZ_MSZIP_COMPATIBLE

This option has been removed in QAT Windows* Release 1.4.

This compression algorithm adds a 4-Byte header to the standard MSzip compressed block:

See Section 6.4.2 regarding the 4-Byte header.

**Note:** MSzip is only able to utilize a 32KB chunk size (hw_buff_sz).

#### 6.4.3 QZ_ZLIB_COMPATIBLE

This option has been removed in QAT Windows* Release 1.4.

This compression algorithm adds a 4-Byte header to the standard Zlib compressed block:

See Section 6.4.2 regarding the 4-Byte header.
6.4.4 QZ_SW_IGZIP

This option has been removed in QAT Windows* Release 1.4.

This supports the ISA-L software igzip solution. The output will be raw Deflate blocks with a 4 Byte header:

Note: QZ_SW_IGZIP is separate from the use case allowing for QAT hardware accelerated Deflate to have a mechanism for software fallback.

6.5 DEFLATE Data Formats

QATzip for Windows* supports multiple data formats using the QZ_DEFLATE compression algorithm.

6.5.1 DEFLATE_RAW

Not available for QAT Windows*.

Used in conjunction with the QZ_DEFLATE compression algorithm, the output will be raw Deflate blocks without any header or tail:

Note: It is up to the user to store the appropriate metadata for decompression, otherwise decompressing the data may result in bad data.

6.5.2 DEFLATE_4B (Windows* Performance)

All releases prior to and including QAT Windows* Release 1.4 has this data format by default and as the only supported data format.

Used in conjunction with the QZ_DEFLATE compression algorithm, the output will be raw Deflate blocks with a 4 Byte header:

The 4 Byte header includes information regarding the length of the current block. The use case for the header is to allow for parallelizable Deflate decompression.

High performance decompression can be achieved on a single buffer using multiple threads, each sending concurrent QAT HW decompress requests. The block headers can easily be traversed due to the header containing the length of the block.
**Note:** It is up to the user to store the appropriate metadata for decompression, otherwise decompressing the data may result in bad data. For example, when decompressing the data, the chunk size/hw_buff_size is used to space out the inflated blocks, if a different hw_buff_size is used to decompress the data then the inflated blocks will either overlap or be spread too far apart, resulting in the data differing from its original compressed form.

### 6.5.2.1 Software Fallback with DEFLATE_4B

All releases prior to and including QAT Windows* Release 1.3 has software fallback only working with this data format.

### 6.5.3 DEFLATE_GZIP

Added in Release 1.4.

Used in conjunction with the QZ_DEFLATE compression algorithm, the output will be raw Deflate blocks with a standard Gzip header:

A file that is compressed using the DEFLATE_GZIP data format is Gzip compliant and can be decompressed with any number of software Gzip decompressors. And in extension, any file that is compressed using Gzip compliant software may be decompressed using the DEFLATE_GZIP data format.

**Note:** Since the current block’s length field is not present in the header, the decompression performance is much lower than DEFLATE_GZIP_EXT or DEFLATE_4B.

### 6.5.4 DEFLATE_GZIP_EXT

Added in Release 1.4.
Used in conjunction with the QZ_DEFLATE compression algorithm, the output will be raw Deflate blocks with a Gzip header that also includes the length of the current block:

| Gzip Ext Header | Compressed Deflate Block |

The resultant Gzip header using this data format will be slightly larger using the DEFLATE_GZIP data format. However, since the current block’s length information is in the Gzip header, the result is that decompression will be much faster than using DEFLATE_GZIP, see Section 6.4.2 for more information.

The advantage of using this data format over the DEFLATE_4B is that the resultant compressed file is Gzip compatible and can be decompressed with any number of software Gzip decompressors.

The disadvantage of using this data format over DEFLATE_4B is that the Gzip header is larger than that of DEFLATE_4B and the overall file size will increase proportionally.
Virtualization with SR-IOV

The QAT devices support virtualization using PCI-Express pass-through or SR-IOV. The Intel® recommended usage to expose QAT devices into Virtual Machines is to use SR-IOV.

Each QAT device supports up to 16 VF’s, with Windows* enumeration starting at 0. Multiple VF’s can be assigned into a single Virtual Machine for added performance or flexibility.

7.1 Platform and Software Considerations

SR-IOV for QAT has been validated on the first- and second-generation Xeon Scalable platforms. The minimum Windows* version required is Windows* Server 2019 for the host. Currently, the only supported VF is Linux* with validation being done on Ubuntu* 16.04 LTS, 18.04 LTS, and 20.04 LTS.

Since the release of Windows* Server 2019, there have been multiple Windows* Server releases via the Semi-Annual Channel. Intel® recommends using Windows* 20H1 or newer for QAT SR-IOV.

Note: When a QAT device is in SR-IOV mode, it may not be used on the Windows* Host Operating System for compression or crypto functionality.

Note: The Intel® Atom® Processor C3000 series does not support SR-IOV.

7.1.1 Windows* QAT SR-IOV Mixed Mode

It is possible to have some Intel® QAT devices with SR-IOV enabled and other devices with QAT disabled. In this situation, the devices with SR-IOV disabled can be used to offload supported compression and crypto workloads on the Host OS while the devices with SR-IOV enabled can have VF devices assigned to Guests.

Note: Although this mode is supported, Intel® has not extensively validated this configuration.

7.1.2 Windows* QAT SR-IOV Software Fallback

The QAT Windows* PF driver supports SR-IOV Software Fallback in the Linux* Guest using the Linux* VF driver when using QAT Engine applications. During Software Fallback, in flight and future requests will be rerouted to software until the VF driver can be brought back online.

The pre-requisite for QAT SR-IOV SW Fallback requires the QAT Virtual Function devices to be removed before doing the SW Fallback event.

Note: The Linux* QAT PF driver does not support SR-IOV SW Fallback. Therefore, the Linux* and Windows* QAT VF driver will not work with SR-IOV SW Fallback with a Linux* QAT PF.
The typical use cases for SR-IOV Software Fallback are for servicing the PF (such as during a driver upgrade) or hot adding and removing VF devices without shutting down the Guest. In addition, it is an extra layer of safeguard should there be a hardware failure.

*Note:* Intel® does not recommend intentionally creating hardware errors.

The recommended minimum Windows* Host version to use with Windows* QAT SR-IOV Software Fallback is Windows* Server 20H1 and newer.

### 7.1.2.1 SR-IOV SW Fallback Limitations

There are currently some limitations to QAT SR-IOV SW Fallback.

1. Limited to Windows* QAT PF (Release 1.5 and newer) and Linux* QAT VF driver (Release L4.13 and newer). The Linux* PF and Linux* VF combination does not support SR-IOV SW Fallback.
2. Only supports QAT Engine applications.
3. The number of QAT VF’s present and configured when the QAT Engine application starts is the maximum number that can be used when adding QAT VF’s.
4. There are limitations when executing multiple concurrent devices add/remove operations on the QAT VF’s.

### 7.2 Enabling SR-IOV on QAT Devices

If the Windows* QAT driver was installed in Hyper-V Mode, then all the Intel® QAT devices should have SR-IOV enabled and ready to go. However, if the Windows* QAT driver was installed in Standalone mode, additional steps are required to enable SR-IOV on the Intel® QAT devices.

SR-IOV can be enabled or disabled on the QAT base driver via the registry. The registry path is of the format as follows:

```
HKLM\SYSTEM\CurrentControlSet\Enum\PCI\DeviceId\Uuid\Device Parameters\Sriov
```

Where the `DeviceId` is the DeviceId of the QAT hardware adapter (e.g. 37C8), and the `Uuid` is the Windows* generated unique identifier of that device. An example path would be as follows:

```
HKLM\SYSTEM\CurrentControlSet\Enum\PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&44b451a&0&00100000\Device Parameters\Sriov
```

In that path, there should be a DWORD entry with name "EnableSriov". By default, the value of that entry is 0, indicating disabled. Changing the value to 1 will enable SR-IOV mode for each QAT device desired for SR-IOV operation. Example using Powershell:

```
Set-ItemProperty -Path "HKLM:\System\CurrentControlSet\Enum\PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&44b451a&0&00100000\Device Parameters\Sriov" -Name "EnableSriov" -Value 1
```

After changing the registry entries, restart all the QAT devices in Device Manager (or via Powershell). As an added option, you may reboot the system.
To confirm SR-IOV is active on the QAT device use the following Powershell cmdlet:

```
Get-VMHostAssignableDevice
```

In the example figure below, there are three QAT devices (VEN_8086&DEV_37C8) that are SR-IOV enabled. Take note of the InstanceId, which will have to be used later on when assigning the QAT VF’s.

```
InstanceID : PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\624A7469D600000000
LocationPath : PCIROOT(34)\PCI(0000)\PCI(0000)\PCI(0000)
CimSession : CimSession: .
ComputerName : AF11-12-WP2-LBG
IsDeleted : False

InstanceID : PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\624A7469D600000000
LocationPath : PCIROOT(34)\PCI(0000)\PCI(0000)\PCI(0000)
CimSession : CimSession: .
ComputerName : AF11-12-WP2-LBG
IsDeleted : False

InstanceID : PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\624A7469D600000000
LocationPath : PCIROOT(34)\PCI(0000)\PCI(0000)\PCI(0000)
CimSession : CimSession: .
ComputerName : AF11-12-WP2-LBG
IsDeleted : False
```

**Note:** Once SR-IOV is enabled on a QAT device, that QAT device will no longer be able to do hardware accelerated compression or cryptographical services on the Host OS.

### 7.3 Guest Creation with QAT Virtual Functions

The following will step through a Virtual Machine creation within Hyper-V as well as assigning the QAT VF’s.

1. Create the Virtual Machine. The following example will create a VM with 4GB memory using “C:\Vhd” as the VHD path.

   ```powershell
   New-VM -Name “MyVM” -MemoryStartupBytes 4GB -VHDPath “C:\Vhd”
   ```

2. Set desired Virtual Machine properties (Virtual CPU’s, network interfaces, etc).
3. Set AutomaticStopAction to Shutdown

   ```powershell
   Get-VM -Name “MyVM” | set-vm -AutomaticStopAction ShutDown
   ```

4. Add QAT VF devices. In this example, we add three QAT VF’s, one from each InstanceId noted in Section 8.2:

   ```powershell
   Get-VM -Name “MyVM” | Add-VMAssignableDevice -InstanceId “PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\624A7469D600000000” -VirtualFunction 0
   ```
Get-VM -Name "MyVM" | Add-VMAssignableDevice -InstancePath "PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&44B451A&0&00100000" - VirtualFunction 0

Get-VM -Name "MyVM" | Add-VMAssignableDevice -InstancePath "PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&86D7F0F&0&00080000" - VirtualFunction 0

5. Start the Virtual Machine.
Get-VM -Name "MyVM" | Start-VM

6. Install the Guest OS of choice.

7.3.1 Example QAT Virtual Function Commands with Powershell

Windows* Server has built-in Powershell cmdlets that facilitate the addition and removal of SR-IOV devices as well as their status.

The following are some examples for adding and removing VF devices.

Note: Use these cmdlets at your own risk.
1. Get all active VF’s from a VM with name "MyVM”:

   Get-VM -Name "MyVM" | Get-VMAssignableDevice

2. Get all active VF’s from a specific InstanceId:

   Get-VM | Get-VMAssignableDevice | Where-Object {$_InstanceId -eq "PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&44B451A&0&00100000"}

3. Remove all QAT VF’s (matching DeviceId) from all VM’s

   Get-VM | Get-VMAssignableDevice | Where-Object {$_InstanceId -match "VEN_8086&DEV_37C8"} | Remove-VMAssignableDevice

4. Remove all QAT VF’s (matching DeviceId) from a VM with name "MyVM”:

   Get-VM -Name "MyVM" | Get-VMAssignableDevice | Where-Object {$_InstanceId -match "VEN_8086&DEV_37C8"} | Remove-VMAssignableDevice

5. Remove all QAT VF from a specific InstanceId from VM with name “MyVM”:

   Get-VM -Name "MyVM" | Get-VMAssignableDevice | Where-Object {$_InstanceId -eq "PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&44B451A&0&00100000"} | Remove-VMAssignableDevice

6. Add QAT Virtual Function #10 from a specific InstanceId to VM with name “MyVM”:

   Get-VM -Name "MyVM" | Add-VMAssignableDevice -InstancePath PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&86D7F0F&0&00080000 - VirtualFunction 10

7.4 QAT Windows* VF Driver Installation

To install the QAT Windows* VF drivers, the process is the same as installing on bare-metal or the QAT Windows* PF driver.
7.5 QAT Linux* VF Driver Installation

This section assumes the Guest OS has been installed and the QAT VF’s have already been attached and are ready for QAT VF driver installation.

Note: More information about QAT Linux* driver could be found on their respective websites detailed in Section 1.4.

Assuming the tarball is located in the home folder (${HOME}):
1. Install the pre-requisite packages (example for Ubuntu* 18.04 LTS given):

   ```
   $ lspcisudo apt update
   $ sudo apt install pciutils g++ pkg-config libssl-dev libudev-dev yasm build-essential zlib1g-dev
   ```

2. Create destination directory for unpacking the QAT Linux* driver:

   ```
   $ mkdir -p ${HOME}/QAT
   ```

3. Unpack the QAT Linux* driver tarball:

   ```
   $ tar -zxf <QatTarball>
   ```
   Where <QatTarball> is the file name of the QAT Linux* driver tarball.

4. (Optional) Restrict access to the unpacked QAT files.

   ```
   $ chmod -R o-rw "${HOME}/QAT/"
   ```

5. Enable SR-IOV build option (user mode driver):

   ```
   $ cd "${HOME}/QAT"
   $ ./configure --enable-icp-sriov=guest
   ```

6. Build and install the driver:

   ```
   $ sudo make install
   ```

7. Verify the QAT VF’s are up and running:

   ```
   $ adf_ctl status
   ```

7.5.1.1 Linux* VF Sample Code

The QAT Linux* driver package ships with a Sample Code that quickly lets the user run hardware accelerated compression and cryptography.

To compile and use the sample code, assuming the same directory structure in the previous section:

1. Compile the Sample Code

   ```
   $ cd ${HOME}/QAT
   $ sudo make samples-install
   ```

2. Run signOfLife tests, which will quickly run through various compression and cryptographical tests on the driver:
7.6 SR-IOV SW Fallback with QAT Engine

An example application that has SR-IOV SW Fallback with the QAT Windows* PF driver and the QAT Linux* VF driver is Async Nginx* with QAT Engine. This section will detail how to test SR-IOV SW Fallback with QAT Engine.

The General Requirements:
1. Root and/or Administrator privileges may be required.
2. Windows* Host is Windows* Server 20H1 or 20H2 with Hyper-V role enabled.
3. QAT Windows* PF driver, Release 1.5 or newer in SR-IOV mode.
4. The Linux* Guest is Ubuntu* 18.04 LTS with Kernel 4.15.
5. QAT Linux* VF driver is Release L4.13 and has been installed as per Section 7.5.
6. Async Nginx* (version 0.45.0 tested; based on Nginx* 1.18.0)
7. QAT Engine (version 0.6.5 tested)
8. OpenSSL* (version 1.1.1j tested)

Please refer to Section 1.4 on where to get the respective project’s source code.

7.6.1 Async Nginx* with QAT Engine Setup

7.6.1.1 Pre-Requisite Packages

The following packages are required:

- autoconfig
- automake
- g++
- gawk
- gcc
- git
- libssl-dev
- libudev-dev
- libtool
- make
- perl
- pkg-config
- udev
- zlib1g-dev

7.6.1.2 Environmental Variables

The following Environmental Variables are assumed, with the home folder as ${HOME}:

```
export QAT_SRC=${HOME}/QAT
export OPENSSL_SRC=${HOME}/openssl
export OPENSSL_INS=${HOME}/openssl-ins
```
7.6.1.3 OpenSSL*/QAT Engine/Async Nginx* Installation

Refer to Section 1.4 for more information on the respective projects and also where to download the software packages.

Build OpenSSL*:

```
cd $OPENSSL_SRC
./config --prefix=$OPENSSL_INS -Wl,-rpath,$LIBRPATH
make
make install
```

Build QAT-Engine:

```
cd $QATENGINE_SRC
./autogen.sh
./configure

   --with-qat_hw_dir=$QAT_SRC
   --with-openssl_install_dir=$OPENSSL_INS
   --enable-qat_debug
   --with-cc-opt=-DQAT_TESTS_LOG

make
make install
```

**Note:** The `--enable-qat_debug` parameter will enable more debugging messages. Note that performance will decline.

Build Async Nginx*:

```
cd $NGINX_SRC
./configure

   --prefix=$NGINX_INS
   --without-http_rewrite_module
   --with-http_ssl_module
   --with-http_stub_status_module
   --with-http_v2_module
   --with-stream
   --with-stream_ssl_module
   --with-debug
   --add-dynamic_module=$NGINX_SRC/modules/nginx_qat_module
   --with-cc-opt="-DNGX_SECURE_MEM -DNGX_INTEL_SDL -"
I$OPENSSL_INS/include -Wnoerror=deprecated-declarations" \ 
--with-ld-opt="-Wl,-rpath=$OPENSSL_INS/lib -L$OPENSSL_INS/lib"

make

make install

Note: The ‘—with-debug’ parameter will enable more debugging messages. Note that
performance will decline.

7.6.1.4 Async Nginx* Configuration

Get the QAT Nginx* sample configuration saved as "$NGINX_INS/conf/nginx.conf":

```
user  root;
worker_processes 1;
error_log  logs/error.log  error;
load_module modules/ngx_ssl_engine_qat_module.so;
ssl_engine {
    use_engine qatengine;
    default_algorithms RSA,EC,DH,DSA;
    qat_engine {
        qat_sw_fallback on;
        qat_offload_mode async;
        qat_notify_mode poll;
        qat_poll_mode internal;
    }
    qat_poll_mode internal;
}

master_process on;
events {
    worker_connections 102400;
}

http {
    include         mime.types;
    default_type    application/octet-stream;
    access_log      off;
    error_log logs/error.log;
    sendfile        on;
    keepalive_timeout 0;
    keepalive_requests 0;
```
server {
    listen       80;
    server_name  localhost;
    access_log off;
    location / {
        root   html;
        index  index.html index.htm;
    }
    error_page  500 502 503 504 /50x.html;
    location = /50x.html {
        root   html;
    }
}

server {
    listen               443 backlog=1023 reuseport;
    server_name          localhost;
    ssl                  on;
    ssl_protocols        SSLv3 TLSv1 TLSv1.1 TLSv1.2;
    ssl_certificate      TestServer.cert.pem;
    ssl_certificate_key  TestServer.key.pem;
    ssl_session_cache    off;
    ssl_session_timeout  5m;
    ssl_asynch           on;
    ssl_buffer_size      64k;
    access_log off;
    ssl_ciphers          ALL;
    ssl_prefer_server_ciphers  off;
    location / {
        root   html;
        index  index.html index.htm;
    }
    location /basic_status {
        stub_status on;
    }
}

Virtualization with SR-IOV
Create and save `TestServer.cert.pem` and `TestServer.key.pem` (the web server certificate used for authentication):

```bash
cd $NGINX_INS/conf/
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout TestServer.key.pem -out TestServer.cert.pem
(openssl ecparam -genkey -out key.pem -name prime256v1
(openssl req -x509 -new -key key.pem -out cert.pem
```

### 7.6.2 Testing SR-IOV SW Fallback

This section details a simple Nginx* loopback test that uses an ECDHE-RSA-AES256-SHA cipher and demonstrating SR-IOV SW Fallback.

In the Linux* Guest, set the QAT VF conf file to use SHIM instead of SSL:

```
[SHIM]
NumberCyInstances = 2
NumberDcInstances = 2
NumProcesses = 1
LimitDevAccess = 0
```

Restart the QAT VF’s for the configuration to be active:

```
adf_ctl restart
```

In the Linux* Guest, start Nginx* service:

```
cd $NGINX_INS
/sbin/nginx -c conf/nginx.conf
```

Start OpenSSL* workload for one minute:

```
cd $OPENSSL_INS/bin
./openssl s_time -connect 127.0.0.1:443 -new -cipher ECDHE-RSA-AES256-SHA -www /1kb-file.txt -time 60
```

On the Host Partition Powershell, hot remove all QAT VF devices, assuming the VM name is “MyVM”:

```
Get-VM -Name MyVM | Get-VMAssignableDevice | Remove-VMAssignableDevice
```

Wait for a few seconds and hot add one QAT VF device back in, assuming the VM name is “MyVM” using the QAT InstancePath from the Section 7.3 and VF 0.
Virtualization with SR-IOV

```
Get-VM -Name MyVM | Add-VMAssignableDevice -InstancePath
PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&86D7F0F&0&00080000 -
VirtualFunction 0
```

### 7.6.3 Verifying SR-IOV SW Fallback

Check QAT Engine logs for the SW Fallback event. The log file is located in /opt/ directory and has a timestamp-based name with base name 'optcmb'. An example timestamp will be used in this section.

```
    cd /opt
    vim optcmb_1623547545.log
```

When the QAT VF’s are removed from the Guest, QAT Engine fallback occurs. there should be the following entries in the log file:

```
[507.473578] Instance: 0 Handle 0x55cfd7759000 Device 0 RESTARTING
[507.473584] Instance: 1 Handle 0x55cfd7759380 Device 0 RESTARTING
Verification of result failed for qat inst_num 0 device_id 0 - fallback
to SW - qat_rsa_decrypt
Resubmitting request to SW - qat_rsa_priv_enc
```

In the example, since we have two crypto instances per VF (NumberCyInstances = 2), there are two 'RESTARTING' events logged by QAT Engine. As indicated, the QAT Engine workload goes to software from this point onwards, until a QAT VF device is added back in.

When the QAT Virtual Functions are added back into the Guest, QAT Engine will resume sending workloads to QAT hardware. There should be the following entries in the log file:

```
[520.476948] Instance: 0 Handle 0x55cfd7759000 Device 0 RESTARTED
[520.476964] Instance: 1 Handle 0x55cfd7759380 Device 0 RESTARTED
```

§
8  Support

Trouble tickets can be submitted to: Resource & Design Center for Development with Intel

For assistance on account setup contact your Intel® representative.