



Windows Technical Guide

Intel® QuickAssist Technology

Hardware Version 1.7/2.0

Production Release

December 2022

CONTENTS

1	Introduction	5
1.1	About This Document	5
1.2	QAT Accelerator Capabilities	5
1.3	Supported Hardware	6
1.4	Supported Operating Systems	6
1.5	Supported Software Versions	7
1.6	Related Documents and References	8
1.7	Terminology	8
2	QAT Platform Considerations	12
2.1	QAT Generations	12
2.2	Environmental Conditions	12
2.3	QAT Device Number	12
2.4	PCI Express Bandwidth	12
2.5	CPU P-States	13
3	QAT Driver Install/Uninstall	14
3.1	Unsupported QAT Driver Install/Uninstall Methods	14
3.2	Windows* QAT Package Installation	14
3.2.1	Installation Without QAT Hardware	14
3.2.2	QatSetup Installer	15
3.2.2.1	QatSetup Installation Modes	15
3.2.3	QAT Install via Command Line	16
3.2.3.1	QatSetup Install via Command Line	16
3.2.3.2	QAT Driver Install via DevCon	17
3.2.3.3	QAT Driver Install via PnPUtil	17
3.2.3.4	DevCon/PnPUtil Installation Considerations	17
3.2.4	QAT Driver Install Summary	18
3.3	QAT Driver Uninstallation	19
3.3.1	QatSetup Uninstall	19
3.3.2	QAT Uninstall via Command Line	19
3.3.2.1	QatSetup Uninstall via Command Line	20
3.3.2.2	QAT Driver Uninstall via DevCon	20
3.3.2.3	QAT Driver Uninstall via PnPUtil	20
3.3.2.4	Devcon/Pnputil Uninstall Considerations	20
3.4	QAT Driver Interaction with the Intel® Chipset Drivers	21
3.5	Windows* QAT Files and Services	21
3.5.1	Windows* QAT Binaries	21
3.5.2	Windows* QAT Services	21
3.5.3	Windows* QATzip Files	22

3.5.4	Intel® ISA-L	22
3.6	Windows* QAT Registry Keys	22
3.6.1	Intel® ISA-L Location	22
3.6.2	Pre-Allocated Memory Sessions	23
3.6.3	SR-IOV Enable	23
4	Windows* QAT Diagnostics	25
4.1	Windows* Event Log Messages	25
4.2	Using Windows* Perfmon Counters for QAT	25
4.2.1	Rate Counters vs. Interval Counters	26
4.3	Windows* Perfmon QAT Counter Descriptions	26
4.3.1	AE Firmware	26
4.3.2	Compression	26
4.3.2.1	Example Compression Counter Usage	27
4.3.3	Cryptography	28
4.3.3.1	ECC	28
4.3.3.2	ECDH	28
4.3.3.3	ECDSA	28
4.3.3.4	RSA	29
5	Windows* QAT Sample Applications	30
5.1	Parcomp	30
5.1.1	Parcomp General Examples	30
5.1.1.1	Parcomp Compressing, Decompressing, and SHA256 Check Using DEFLATE_RAW	30
5.1.1.2	Parcomp Compressing and Decompressing Using DEFLATE_GZIPEXT	31
5.1.1.3	Parcomp Compressing and Decompressing with Multiple Independent Threads	32
5.1.2	Parcomp Software Fallback Using ISA-L	33
5.1.2.1	Parcomp Software Threshold Example	33
5.1.2.2	Parcomp Software Fallback Example	33
5.2	Cngtest	34
5.2.1	CNG Scaling	34
5.2.2	Cngtest General Examples	34
5.2.3	Cngtest Fallback	35
6	QATzip API Guide	36
6.1	QATzip Limitations	36
6.2	Using QATzip Without QAT Hardware	37
6.3	QATzip API Function Overview	37
6.3.1	Windows* and Linux* QATzip Differences	37
6.3.2	qzInit/qzClose Overview	37
6.3.3	qzSessionSetup/qzTeardownSession Overview	38
6.3.4	qzCompress/qzDecompress Overview	39
6.3.5	qzCompressExt/qzDecompressExt Overview	39
6.3.5.1	QATzip Extended Return Codes	39
6.3.5.2	qzCompressCrc64Ext/qzDecompressCrc64Ext	40
6.3.6	qzCompressCrc64/qzDecompressCrc64	40
6.3.6.1	qzGetSessionCrc64Config	40
6.3.7	getQatVersionIndex Overview	40
6.4	Compression Algorithms	41
6.4.1	QZ_DEFLATE	41
6.5	DEFLATE Data Formats	41
6.5.1	DEFLATE_RAW	41
6.5.2	DEFLATE_4B (Windows* Performance)	41
6.5.3	DEFLATE_GZIP	42

6.5.4	DEFLATE_GZIP_EXT	43
7	Virtualization with SR-IOV	44
7.1	Platform and Software Considerations	44
7.1.1	PF/VF Software Compatibility	44
7.1.1.1	PF/VF Software Compatibility QAT HW 2.0	45
7.1.1.2	PF/VF Software Compatibility QAT HW 1.7	45
7.1.2	QAT VF Device Scaling	46
7.1.3	Windows* QAT SR-IOV Mixed Mode	46
7.1.4	Windows* QAT SR-IOV Software Fallback	46
7.1.4.1	SR-IOV Software Fallback Limitations	46
7.2	Enabling SR-IOV on QAT Devices	47
7.3	Guest Creation with QAT Virtual Functions	48
7.3.1	Example QAT Virtual Function Commands with PowerShell	48
7.4	QAT Windows* VF Driver Installation	49
7.5	QAT Linux* VF Driver Installation	50
7.6	Linux* VF Sample Code	50
7.7	SR-IOV Software Fallback with QAT Engine	51
7.7.1	Async Nginx* with QAT Engine Setup	51
7.7.1.1	Pre-Requisite Packages	51
7.7.1.2	Environmental Variables	52
7.7.1.3	OpenSSL*/QAT Engine/Async Nginx* Installation	52
7.7.1.4	Async Nginx* Configuration	53
7.7.2	Testing SR-IOV Software Fallback	54
7.7.3	Verifying SR-IOV Software Fallback	55
8	Support	56

Intel® QuickAssist for Windows* Technical Guide**Version: W.2.0.1****Revision W.2.0.1**

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.

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 technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer.

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 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.

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

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548- 4725 or visit www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Copyright © 2022, Intel Corporation. All rights reserved.

Table 1:: Revision History

Document Number	Revision Number	Description	Revision Date
648260	2.0.1	<ul style="list-style-type: none"> • Updated Supported Hardware • Updated Supported OS's • Updated Supported Software Versions • Updated PCI Express Bandwidth • Updated CPU P-States • Updated Windows* QAT Binaries • Updated QATzip Limitations • Updated qzCompressCrc64/ qzDecompressCrc64 • Added PF/VF Software Compatibility QAT HW 2.0 • Added PF/VF Software Compatibility QAT HW 1.7 • Removed references to driver release 1.4.0 and older 	December 2022

continues on next page

Table 1 – continued from previous page

Document Number	Revision Number	Description	Revision Date
648260	1.9.0	<ul style="list-style-type: none"> • Updated Section 1.5 Supported Software Versions • Updated Section 3.1 Unsupported QAT Driver Install Methods • Rename Section 3.2.2.1 to Installation Modes 	October 2022
648260	1.8.0	<ul style="list-style-type: none"> • Added Section 1.2 QAT Accelerator Capabilities • Updated Section 1.4 Supported Software Versions • Added Section 2.1 Environmental Conditions • Updated Section 3.5.4 Intel® ISA-L • Updated Section 5.2.2 Cngtest General Examples • Updated Section 6.2 Using QATzip without QAT Hardware • Updated Section 6.3.2 qzInit Overview • Updated Section 6.3.3 qzSessionSetup Overview • Added Section 6.3.6 getQatVersionIndex Overview • Updated Section 7.1.1 PF/VF Software Compatibility 	June 2022

continues on next page

Table 1 – continued from previous page

Document Number	Revision Number	Description	Revision Date
648260	1.7.0	<ul style="list-style-type: none"> • Updated Section 1 Introduction • Updated Section 1.3 Supported Operating Systems • Updated Table 2. Supported Guest Operating Systems for SR-IOV • Updated Section 1.4 Supported Software Versions • Added Note to Section 2.1 QAT Device Number • Removed Note in Section 3.2.2.1 • Added Section 3.24 QAT Driver Install Summary • Updated Section 3.2.3.1 QatSetup Install via Command Line • Updated Section 3.5.1 Windows* QAT Binaries • Updated Section 3.5.2 Windows* QAT Services • Updated Section 3.5.3 Windows* QATzip Files • Updated Section 3.5.4 Intel® ISA-L • Added Section 3.6 Windows* QAT Registry Keys • Added Section 4.1 Windows* Event Log Messages • Updated Section 6.1 Limitations • Moved Section 6.2.1 to Section 6.3.5.1 QATzip Extended Return Codes • Added Section 6.3 QATzip API Function Overview • Added Section 6.3.1 Windows* and Linux* QATzip Differences • Added Section 6.3.2 qzInit Overview • Added Section 6.3.3 qzSessionSetup Overview • Added Section 6.3.4 qzCompress/qzDecompress Overview • Added Section 6.3.5 qzCompressExt/ qzDecompressExt Overview • Updated Section 7.1 SR-IOV Platform and Software Considerations • Added Section 7.1.1 PF/VF Software Compatibility • Added Section 7.1.2 QAT VF Device Scaling • Updated Section 7.1.4 Windows* QAT SR-IOV Software Fallback • Added Note to Section 7.4 Windows* VF Driver Installation 	March 2022
648260	1.5.0	<ul style="list-style-type: none"> • Updated Table 2. Supported Guest Operating Systems for SR-IOV • Updated 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

continues on next page

Table 1 – continued from previous page

Document Number	Revision Number	Description	Revision Date
648260	1.4.0	<ul style="list-style-type: none">• Initial release	August 2021

INTRODUCTION

This document discusses the technical details related to using the following Intel® QuickAssist Technology (QAT) drivers and software:

- Release W.1.9.0-0008 for Intel® QAT HW 1.7
- Release W.2.0.1-0016 for Intel® QAT HW 2.0

1.1 About This Document

The target audience for this guide are as follows:

- System Administrators interested in utilizing or deploying hardware accelerated compression and cryptography on Windows* with either Windows* or Linux* Guests.
- Application developers who want to use Intel® QuickAssist to accelerate compression and cryptography workloads on Windows*.

1.2 QAT Accelerator Capabilities

The Intel® QuickAssist device can accelerate certain compression/decompression and cryptography workloads. This may free the host's CPU resources to be utilized for other compute tasks.

Compression/Decompression

Windows QAT supports hardware accelerated compression and decompression via the Intel* QATzip API in user mode only. It supports the following offloads:

- DEFLATE based compression algorithm. The supported data formats are:
 - Raw Deflate with 4-Byte length header
 - Gzip as defined in RFC 1952
 - Gzip Extended (Gzip with additional metadata in the header)

Cryptography

Windows QAT supports hardware accelerated Public Key Encryption via the Microsoft* Cryptography Next Generation (CNG) API in user mode only. It supports the following offloads:

- ECDH using curves nistP256, nistP384, nistP521, and Curve25519
- ECDSA using curves nistP256, nistP384, nistP521
- RSA with key lengths of 2048, 3072, 4096 bits

Windows QAT also supports hardware accelerated symmetric crypto via Microsoft* Cryptography Next Generation (CNG) API in kernel mode only. It supports the following offloads:

- AES (CBC, CCM, GCM, XTS with 256 bit key length)

1.3 Supported Hardware

Intel® QuickAssist Technology (QAT) Release W.2.0.1-0016 supports the following hardware:

- Intel® Xeon® Scalable Processor with Intel® QAT Gen4 in 1-Socket and 2-Socket configurations.

Note: Intel® Xeon® Scalable Processor with Intel® QAT Gen4 in 4-Socket and 8-Socket configurations may work, but may not be optimized.

Intel® QuickAssist Technology (QAT) Release W.1.9.0-0008 supports the following hardware:

- Intel® QuickAssist Adapter 8960 and 8970
- Intel® Xeon® Scalable First-, Second-, and 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)

1.4 Supported Operating Systems

The following table illustrates the supported Host Operating Systems:

Table 1.1:: Validated Host Operating Systems

Host Operating System	Intel® QAT 8960/8970/C62x	Intel® QAT C3000	Intel® QAT 4xxx
Windows* Server 2016	Yes	No	No
Windows* Server 2019	Yes	No	No
Windows* Server 2022	Yes	No	Yes
Windows* 10 Enterprise LTSC	No	Yes	No

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

Table 1.2:: Validated Guest Operating Systems

Guest Operating System	Intel® QAT C62x Virtual Function	Intel® QAT 4xxx Virtual Function
Windows* Server 2016	Full QAT HW/SW Support	No
Windows* Server 2019	Full QAT HW/SW Support	Full QAT HW/SW Support
Windows* Server 2022	Full QAT HW/SW Support	Full QAT HW/SW Support
Windows* 10 Enterprise 21H2	SW ISA-L / MS SQL Restore only	SW ISA-L / MS SQL Restore only
Windows* 11 Enterprise 21H2	SW ISA-L / MS SQL Restore only	SW ISA-L / MS SQL Restore only
Ubuntu* 18.04 LTS, Kernel 4.15	Full QAT HW/SW Support	Full QAT HW/SW Support
Ubuntu* 20.04 LTS, Kernel 5.4	Full QAT HW/SW Support	Full QAT HW/SW Support

Note: Intel® recommends updating Windows* to the latest Cumulative Update.

Note: Windows 10 and Windows 11 Enterprise has only been validated for software ISA-L support specifically for Microsoft* SQL software restore from QAT hardware or ISA-L software backup.

Important: Other Host/Guest Operating System combinations may work but has not been validated by Intel®.

1.5 Supported Software Versions

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

Table 1.3:: Validated Software Versions

Software Name	Package Version	Description
Async-Mode Nginx*	0.4.7	SR-IOV QAT-Engine Software Fallback Nginx*
Intel® ISA-L	2.30	Windows* QAT Compression Software Fallback
Intel® Linux* QAT Drivers	L4.19-00005	QAT Linux* HW 1.7 Driver (for SR-IOV)
Intel® Linux* QAT Drivers	L1.0.0-00017	QAT Linux* HW 2.0 Driver (for SR-IOV)
Intel® QAT Engine	v0.6.12	SR-IOV QAT-Engine Software Fallback
OpenSSL*	1.1.1n	SR-IOV QAT-Engine Software Fallback
Windows* Server 2016	KB5016622	Windows* Cumulative Updates to August 2022
Windows* Server 2019	KB5016623	Windows* Cumulative Updates to August 2022
Windows* Server 2022	KB5016627	Windows* Cumulative Updates to August 2022

1.6 Related Documents and References

Async Mode for Nginx

GZIP File Format Specifications RFC1952

Intel® Intelligent Storage Acceleration Library

Intel® QuickAssist Technology API Programmer's Guide

Intel® QuickAssist Technology OpenSSL* Engine

Intel® QuickAssist Technology QATzip

Intel® QuickAssist Technology Software for Linux* Drivers (Hardware Version 1.7)

Intel® QuickAssist Technology Software for Linux* Getting Started Guide (Hardware Version 1.7)

Intel® QuickAssist Technology Software for Linux* Release Notes (Hardware Version 1.7)

Microsoft* Cryptography API Next Generation

Microsoft* DevCon GitHub

Microsoft* PowerShell GitHub

OpenSSL Cryptography and SSL/TLS Toolkit

1.7 Terminology

ADF

Acceleration Driver Framework.

AEAD

Authenticated Encryption With Associated Data.

AES

Advanced Encryption Standard.

API

Application Programming Interface.

ASIC

Application Specific Integrated Circuit.

BDF

Bus Device Function.

BIOS

Basic Input/Output System.

BOM

Bill of Materials.

BSD

Berkeley Software Distribution.

CBC

Cipher Block Chaining mode.

CCM

Counter with CBC-MAC mode.

CLI	Command Line Interface.
CnV	Compress and Verify.
CnVnR	Compress and Verify and Recover.
C-States	C-States are advanced CPU current lowering technologies.
CY	Cryptography.
DC	Data Compression.
DID	Device ID.
DMA	Direct Memory Access.
DRAM	Dynamic Random Access Memory.
DSA	Digital Signature Algorithm.
DTLS	Datagram Transport Layer Security.
ECC	Elliptic Curve Cryptography.
ECDH	Elliptic Curve Diffie-Hellman.
FLR	Function Level Reset.
FW	Firmware.
GCM	Galois/Counter Mode.
GPL	General Public License.
GUI	Graphical User Interface.
HMAC	Hash-based Message Authentication Mode.
IA	Intel® Architecture.
IEEE	Institute of Electrical and Electronics Engineers.

IKE

Internet Key Exchange.

Intel® ISA-L

Intel® Intelligent Storage Acceleration Library. This includes an optimized library for fast software Deflate compression and decompression.

Intel® QAT

Intel® QuickAssist Technology.

Intel® SpeedStep® Technology

Advanced means of enabling very high performance while also meeting the power-conservation needs of mobile systems.

Intel® VT

Intel® Virtualization Technology.

IOCTL

Input Output Control function.

IOMMU

Input-Output Memory Management Unit.

LAC

LookAside Crypto.

Latency

The time between the submission of an operation via the QuickAssist API and the completion of that operation.

MSI

Message Signaled Interrupts.

NUMA

Non-uniform Memory Access.

Offload Cost

This refers to the cost, in CPU cycles, of driving the hardware accelerator. This cost includes the cost of submitting an operation via the Intel® QuickAssist API and the cost of processing responses from the hardware.

OS

Operating System.

PCH

Platform Controller Hub. In this manual, a Platform Controller Hub device includes standard interfaces and Intel® QAT Endpoint and I/O interfaces.

PCI

Peripheral Component Interconnect.

PF

Physical Function.

PKE

Public Key Encryption.

PowerShell

Cross-platform command-line shell and scripting language using the .NET Common Runtime.

RAS

Reliability, Availability, Serviceability.

RSA

Rivest-Shamir-Adleman.

SAL

Service Access Layer.

SGL

Scatter-Gather List.

SHA

Secure Hash Algorithm.

sIOV

Intel® Scalable I/O Virtualization

SoC

System-on-a-Chip.

SR-IOV

Single-Root Input/Output Virtualization.

SSL

Secure Sockets Layer.

SYM

Symmetric Crypto.

TCG

Trusted Computing Group.

Throughput

The accelerator throughput usually expressed in terms of either requests per second or bytes per second.

TLS

Transport Layer Security.

TPM

Trusted Platform Module.

UDP

User Datagram Protocol.

USDM

User Space DMA-able Memory.

VF

Virtual Function.

VHD

Virtual Hard Disk, VHD(x) is the successor file format.

VM

Virtual Machine.

WDK

Windows* Driver Kit

WPP

Windows* Software Trace Pre-processor

QAT PLATFORM CONSIDERATIONS

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

2.1 QAT Generations

It is not recommended to mix-match QAT generations on the same system. The W.2.0.1 will not have QAT HW 1.7 drivers and the W.1.9.0-0008 will not have QAT HW 2.0 drivers.

2.2 Environmental Conditions

The QAT device (on SoC, chipset, or PCI-Express add-in adapter) must be operating at temperatures defined in the Thermal specifications for the device. Failure to provide for adequate cooling may cause system instability when performing QAT accelerated compression or cryptography operations.

2.3 QAT Device Number

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

Important: For the Intel® QAT device driver to load properly, physical memory is required in the NUMA socket that the QAT device is installed in.

2.4 PCI Express Bandwidth

For optimal performance, the PCI-Express based QAT HW 1.7 add-in adapter should have 16x PCI Express 3.0 lanes. For QAT HW 1.7 using the Intel® C62x chipset, the system should have 8x PCI Express 3.0 lanes per QAT device. Using anything less may result in performance degradation in I/O bound operations, such as compression and decompression.

2.5 CPU P-States

Enabling CPU P-States may lead to a noticeable drop in compression or decompression 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 be limited to the Intel® Xeon® Scalable First-, Second-, Third-, and Fourth-generation platforms.

QAT DRIVER INSTALL/UNINSTALL

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.

Important: 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 may lead to undesirable behavior.

Note: For more information, see [PowerShell Working with Software Installations](#)

3.2 Windows* QAT Package Installation

All the examples in this section assume 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

The 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 QatSetup Installation Modes

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

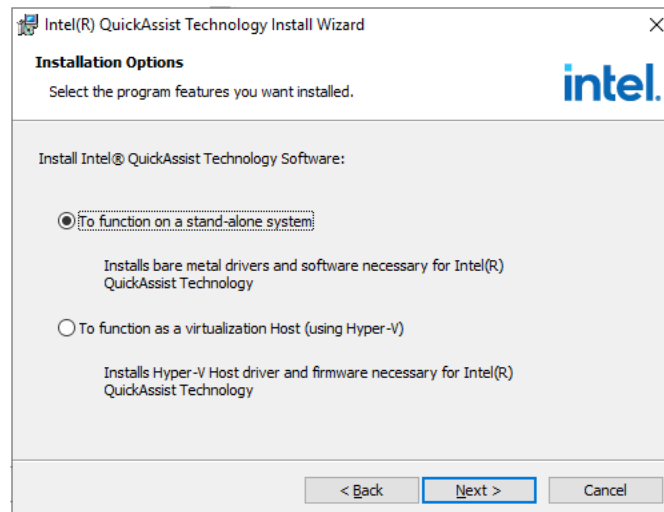


Figure 3.1:: Installation Options

Standalone Mode

The stand-alone mode 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).

Hyper-V Mode

The Hyper-V mode will only install the QAT Windows* base driver. 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.

DLL Only Mode

The DLL Only Mode is only available if no supported Intel® QAT Hardware is detected on the target system. A warning should appear and upon selecting the affirmative, only the *qatzip.dll* and *isa-l.dll* will be installed on the target system.

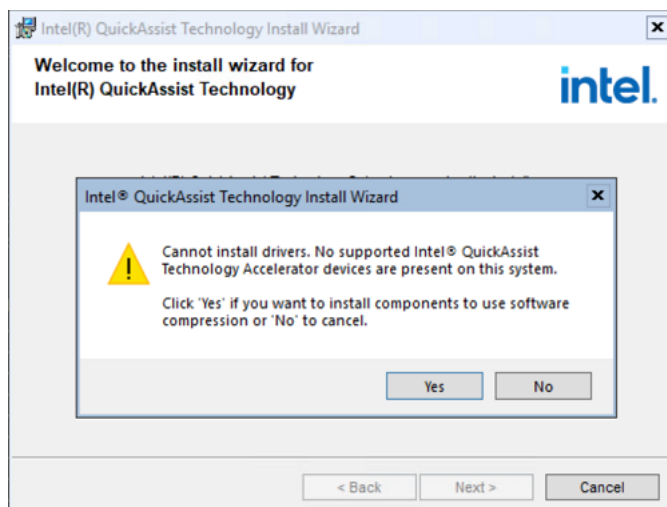


Figure 3.2:: DLL Mode Warning

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.

Important: 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
```

An example silent, no touch, Hyper-V Mode installation using QatSetup.exe would be:

```
QatSetup.exe /passive /qn HYPERVMODE=1
```

An example silent, no touch, no QAT drivers/services installation using QatSetup.exe would be:

```
QatSetup.exe /passive /qn INSTALLDLLONLY=1
```

Note: HYPERVMODE=1 installation will fail if the Windows* OS is Windows* Server 2016 or older or any version of Windows* client.

Note: When attempting to install the Windows* QAT driver package on systems without QAT hardware via command line, INSTALLDLLONLY=1 flag must be used. Otherwise, the installation will fail.

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. Create 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 compression services are 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 user mode crypto service (if cryptography services are to be used):

```
sc.exe create cpmprovuser type= kernel start= boot binpath=
C:\\Windows\\System32\\drivers\\CPMProvUser.sys
sc.exe start cpmprovuser
```

5. Ensure all the requisite services can start successfully.
6. Start the QAT device driver via Device Manager or PowerShell. The device status should be 'OK'.

3.2.4 QAT Driver Install Summary

Starting with Release 1.7, after a successful QAT driver package install, a text file named *QATInstallSummary.log* will be generated at the path '*C:\\Program Files\\Intel\\Intel(R) QuickAssist Technology*'. This file will detail the specific Windows* QAT components and services that were installed.

Additionally, the last dialog box for the QAT driver installation package will feature an option to display the installation summary.

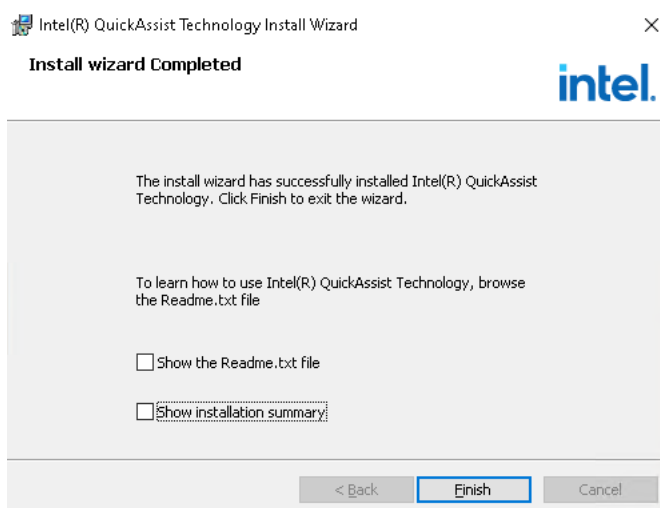


Figure 3.3:: Show Readme and Installation Summary

3.3 QAT Driver Uninstallation

All the examples in this section assume 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 the “Uninstall a Program” submenu. Select the entry with Name “*Intel® QuickAssist Technology*”.

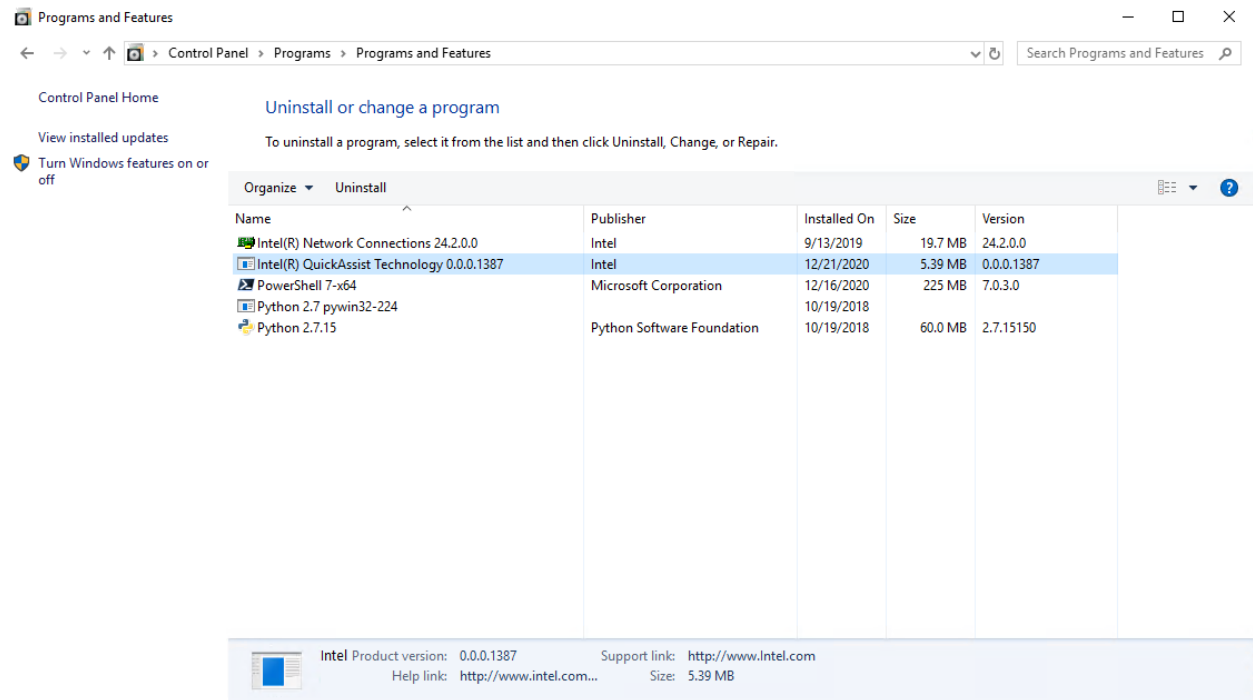


Figure 3.4:: QatSetup Uninstall

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.

Important: 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; otherwise, it may lead to undesirable behavior.

3.3.2.1 QatSetup Uninstall via Command Line

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

```
QatSetup.exe --help
```

An example of a silent, no touch uninstall using QatSetup.exe would be:

```
QatSetup.exe /x /qn /norestart
```

3.3.2.2 QAT Driver Uninstall via DevCon

DevCon is a Windows* command line application, for more information, see [DevCon/PnPUtil Installation Considerations](#).

Example DevCon command to uninstall the QAT driver with DeviceId '37C8':

```
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. The PnPUtil uninstall functionality requires Windows* version 1607 or later.

Example PnPUtil command to uninstall the QAT driver where oem7.inf is the inf file generated when installing the QAT driver:

```
pnputil.exe /delete-driver oem7.inf /uninstall /force /reboot
```

3.3.2.4 Devcon/Pnputil Uninstall 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. (Recommended) 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 contain 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:

- In the registry entry for the QAT device, there may be double the UUID entries.
- 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.
- 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 the System32 Drivers folder:

- CfQat - Windows* QAT binary used for QAT hardware accelerated compression and decompression.
- CPMProv - Windows* QAT binary for QAT hardware accelerated cryptography offloads in kernel mode.
- CPMProvUser - Windows* QAT binary for QAT hardware accelerated cryptography offloads in user mode.
- icp_qat - QAT Windows* base driver file.

3.5.2 Windows* QAT Services

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

- CfQat
- CPMProv (QAT HW 2.0 and later only)
- 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 and copied to 'C:\Program Files\Intel\Intel(R) QuickAssist Technology\CompressionLibrary':

- intelisa-l.win-x64.2.30.0.nupkg - The ISA-L NuGet package installed
- libqatzip.lib - The Windows* QATzip static library
- qatzip.lib - The Windows* QATzip DLL library file to hook into the DLL
- qatzip.h - The Windows* QATzip header file
- pdb\libqatzip.pdb - The symbol file for libqatzip.lib
- pdb\qatzip.pdb - The symbol file for qatzip.dll

The QATzip files installed in the System32 directory: - qatzip.dll - The Windows* QATzip Dynamic Link Library.

3.5.4 Intel® ISA-L

Starting with QAT Windows* Release 1.8, the QAT installer package will attempt to install the Intel® ISA-L DLL regardless of whether the NuGet package manager is available.

For information regarding the ISA-L Location registry key, refer to [Intel® ISA-L Location](#).

3.6 Windows* QAT Registry Keys

There are several notable Windows* QAT registry keys to help the end-user customize the Windows* QAT driver for their use case.

3.6.1 Intel® ISA-L Location

Registry Key Location:

HKLM\SOFTWARE\Intel\QAT\SWFallback\Location

Registry Key Default Value:

C:\Program Files\Intel\ISA-L\intelisa-l.win-x64.2.30.0\runtimes\win-x86\native

Registry Key Description:

Windows* QATzip compression will attempt to use the *isa-l.dll* specified at the *Location* value defined by this registry key.

The *Location* value can be modified to change the search path of *isa-l.dll*. The default *Location* value is based on the ISA-L version.

Note: If the ISA-L Location registry key is not present, QATzip will attempt to look for ISA-L in the system32 directory.

3.6.2 Pre-Allocated Memory Sessions

Registry Key Location:

HKLM\SYSTEM\CurrentControlSet\Services\cfQat\Parameters\PreAllocatedMemorySessions

Registry Key Default Value:

4

Registry Key Description:

Allows the CfQat service to pre-allocate contiguous memory to use as a cache for smaller compression requests. This only works when using QATzip with chunk size of 64KB. This is useful if the system is already using large amounts of system memory (which results in slow contiguous memory allocations).

The default value of 4 approximately translates into an additional 130MB of pre-allocated contiguous memory. A value of 0 means that this feature is disabled. The maximum value is 16 and any value not between 0-16 will translate into the default value.

Note: A restart of the CfQat service is required to propagate any new settings for *PreAllocatedMemorySessions*.

3.6.3 SR-IOV Enable

Registry Key Location:

HKLM\SYSTEM\CurrentControlSet\Enum\PCNDeviceId\Uuid\Device Parameters\Sriov

Where *DeviceId* is the Device ID of the QAT hardware adapter (e.g., 37C8).

Where *Uuid* is the Windows* generated unique identifier of that device.

An example registry path would be:

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

Registry Key Default Value:

0 or 1

The default value is 0 if the QAT driver installation was done in Standalone mode. If the QAT driver installation was done in Hyper-V Mode, then the default value is 1.

Registry Key Description:

The *EnableSriov* registry key determines if the QAT device is in SR-IOV mode.

A value of 1 indicates that the QAT device is in SR-IOV. A value of 0 indicates that the QAT device is not in SR-IOV mode. The value is per QAT device.

Note: A restart of the QAT device(s) are required to propagate any new settings for *EnableSriov*.

WINDOWS* QAT DIAGNOSTICS

4.1 Windows* Event Log Messages

The following Windows* Events are available for the QAT driver.

Windows* Events Available for the QAT Driver

Table 4.1:: Windows* Events for the QAT Driver

Event ID	Min Release	Level	Description
64	W.1.7.0	Warning	The QAT device is going to reset.
65	W.1.7.0	Warning	The QAT device has been reset.
96	W.1.7.0	Error	The QAT device has experienced a failure and will reset.

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 utilize QAT.

The AE firmware counters are available in Windows* native OS operation and in the Windows* Host OS partition when running Hyper-V.

All other Windows* counters (excluding the above AE 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 unused counters in the manifest file.

4.2.1 Rate Counters vs. Interval Counters

There are two main types of counters 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 1,000 bytes of data is compressed in the 1 second interval, the throughput will be reported as 1,000 in the Compression Throughput counter and is interpreted as 1,000 bytes per second for that interval. The Compression Bytes In per Interval counter will also report 1,000 and is interpreted as 1,000 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 1,000 bytes per second, the Compression Throughput counter will report 1,000 bytes per second because there was 5,000 bytes compressed divided by 5 seconds for a rate of 1,000 bytes per second. The Compression Bytes In per Interval counter will read 5,000 because there was 5,000 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 4.2:: AE Firmware Counters

Counter	Description
Request Retries	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.
AE Firmware Responses	Responses per second from QAT for all responses for the ring bundle.
AE Firmware Requests	Requests per second to QAT for all requests for the ring bundle.

4.3.2 Compression

The following counters are available for QAT hardware compression and decompression. They can be found in the *Intel QuickAssist Compression* counters section.

Table 4.3:: QAT Hardware Compression and Decompression Counters

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

4.3.2.1 Example Compression Counter Usage

Example of interval counter gathering during Calgary corpus compression. Observe that the resultant value is the exact length of the Calgary corpus.

```
Get-Counter -Counter "\Intel QuickAssist Accelerator Compression(*)\`
  Compression Bytes In per Interval" -SampleInterval 5 -MaxSamples 1
```

```

PS C:\Users\Administrator> Get-Counter -Counter $counter -SampleInterval 5 -MaxSamples 1^C
PS C:\Users\Administrator> cd c:\
PS C:\>
PS C:\>
PS C:\> Get-Counter -Counter $counter -SampleInterval 5 -MaxSamples 1

Timestamp                CounterSamples
-----
1/7/2021 12:01:53 PM      \\af11-12-wp2-lbg\intel quickassist accelerator compression(_total)\compression bytes in per
                             interval :
                             3251493

```

Figure 4.1:: Windows* QAT Counters Compression Example

4.3.3 Cryptography

The following counters are available for QAT hardware cryptography. They can be found in the *Intel QuickAssist Accelerator ECC* counters section.

4.3.3.1 ECC

The following counters are available for QAT hardware ECC cryptography.

Table 4.4:: QAT Hardware ECC Crypto Counters

Counter	Description
ECC Point Verify Errors/sec	ECC Point Verify errors returned to QAT driver.
ECC Point Multiple Errors/sec	ECC Point Multiply errors returned to QAT driver.
ECC Point Verify Ops/sec	ECC Point Verify successful operations per second.
ECC Point Multiple Ops/sec	ECC Point Multiply successful operations per second.

4.3.3.2 ECDH

The following counters are available for QAT hardware ECDH cryptography. They can be found in the *Intel QuickAssist Accelerator ECDH* counters section.

Table 4.5:: QAT Hardware ECDH Crypto Counters

Counter	Description
ECDH Point Multiple Errors/sec	ECDH Point Multiply errors returned to QAT driver.
ECDH Point Multiply Ops/sec	ECDH Point Multiple successful operations per second.

4.3.3.3 ECDSA

The following counters are available for QAT hardware ECDSA cryptography. They can be found in the *Intel QuickAssist Accelerator ECDSA* counters section.

Table 4.6:: QAT Hardware ECDSA Crypto Counters

Counter	Description
ECDSA Verify Errors/sec	ECDSA Verify errors returned to QAT driver.
ECDSA Sign RS Errors/sec	ECDSA Signature RS errors returned to QAT driver.
ECDSA Sign S Errors/sec	ECDSA Signature S errors returned to QAT driver.
ECDSA Sign R Errors/sec	ECDSA Signature R errors returned to QAT driver.
ECDSA Verify Operations/sec	ECDSA Verify successful operations per second.
ECDSA Sign RS Operations/sec	ECDSA Signature RS successful operations per second.
ECDSA Sign S Operations/sec	ECDSA Signature S successful operations per second.
ECDSA Sign R Operations/sec	ECDSA Signature R successful operations per second.

4.3.3.4 RSA

The following counters are available for QAT hardware RSA cryptography. They can be found in the *Intel QuickAssist Accelerator RSA* counters section.

Table 4.7:: QAT Hardware RSA Crypto Counters

Counter	Description
RSA Key Generation Errors/sec	RSA Key Generation errors returned to QAT driver.
RSA Encrypt Errors/sec	RSA Encrypt errors returned to QAT driver.
RSA Decrypt Errors/sec	RSA Decrypt errors returned to QAT driver.
RSA Key Generation Ops/sec	RSA Key Generation successful operations per second.
RSA Encrypt Operations/sec	RSA Encrypt successful operations per second.
RSA Decrypt Operations/sec	RSA Decrypt successful operations per second.

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(R) 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.

Important: 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 corpus into 'calgary_compressed' with DEFLATE_4B, Level 1, 64KB Chunk Size:

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

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> .\parcomp.exe -i C:\temp\calgary -o c:\temp\calgary_compressed -p qat -l 1 -c 64
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\temp\calgary (3251493 Bytes)
Writing output file: c:\temp\calgary_compressed (1270715 Bytes)

Deflation Ratio (%age)      : 39.1
Thruput (uncompressed Mbps): 11262.044
Time (ms)                   : 1.653

Note:- All times exclude file I/O and are measured around the call to the
qzCompress() API only.
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression>

```

Decompressing 'calgary_compressed' to 'calgary_decompressed':

```
parcomp.exe -i C:\temp\calgary_compressed -o C:\temp\calgary_decompressed -p qat -c 64 -d
```

Using PowerShell to verify the decompressed file against original with SHA256 hash:

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

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> .\parcomp.exe -i C:\temp\calgary_compressed -o c:\temp\calgary_decompressed -p qat -c 64 -d
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\temp\calgary_compressed (1270715 Bytes)
Writing output file: c:\temp\calgary_decompressed (3251493 Bytes)

Inflation Ratio (%age)      : 255.9
Thruput (uncompressed Mbps): 17696.404
Time (ms)                   : 0.948

Note:- All times exclude file I/O and are measured around the call to the
qzDecompress() API only.
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> (Get-FileHash C:\temp\calgary_decompressed -Algorithm SHA256).Hash -eq (Get-FileHash C:\temp\calgary -Algorithm SHA256).Hash
True
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression>

```

5.1.1.2 Parcomp Compressing and Decompressing Using DEFLATE_GZIPEXT

Compressing Calgary corpus 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 qatgziptext -d
```

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> .\parcomp.exe -i C:\temp\calgary -o c:\temp\calgary_compressed -p qatzipext -l 1 -c 64
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\temp\calgary (3251493 Bytes)
Writing output file: c:\temp\calgary_compressed (1272115 Bytes)

Deflation Ratio (%age)      : 39.1
Thruput (uncompressed Mbps): 12068.828
Time (ms)                   : 1.503

Note:- All times exclude file I/O and are measured around the call to the
qzCompress() API only.
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> .\parcomp.exe -i C:\temp\calgary_compressed -o c:\temp\calgary_decompressed -p qatzipext -d
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\temp\calgary_compressed (1272115 Bytes)
Writing output file: c:\temp\calgary_decompressed (3251493 Bytes)

Inflation Ratio (%age)      : 255.6
Thruput (uncompressed Mbps): 17044.718
Time (ms)                   : 0.992

Note:- All times exclude file I/O and are measured around the call to the
qzDecompress() API only.
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression>

```

5.1.1.3 Parcomp Compressing and Decompressing with Multiple Independent Threads

For optimal performance, the parcomp sample application can 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 -l 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.

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> .\parcomp.exe -i C:\temp\silesia -o c:\temp\silesia_compressed -p qat -l 1 -c 64 -k 4096 -Q -t 9 -j 60 -n 100
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\temp\silesia (211938580 Bytes)
All threads completed as Expected.

Deflation Ratio (%age)      : 37.8
Thruput (uncompressed Mbps): 72882.634
Processing Block size       : 4096 KB
Block count                 : 51
Time/block (ms)             : 4.105

Note:- All times exclude file I/O and are measured around the call to the
qzCompress() API only.
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression> .\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
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\temp\silesia_compressed0 (80147381 Bytes)
All threads completed as Expected.

Inflation Ratio (%age)      : 264.4
Thruput (uncompressed Mbps): 124910.374
Processing Block size       : 4096 KB
Block count                 : 51
Time/block (ms)             : 2.395

Note:- All times exclude file I/O and are measured around the call to the
qzDecompress() API only.
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Compression>

```

5.1.2 Parcomp Software Fallback Using ISA-L

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

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

5.1.2.1 Parcomp Software 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_4B, Level 1, 64KB Chunk Size/512KB Block Size with threshold of 256KB:

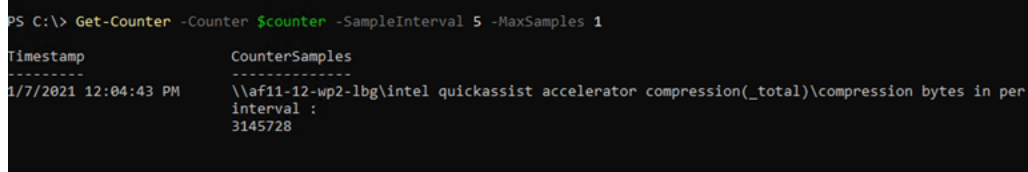
```
parcomp.exe -i C:\temp\calgary -o C:\temp\calgary_compressed -p qat -l 1 -c 64 -k 512 -FT 256
```

The Calgary corpus is 3,251,493-byte length. Since the block size is 512KB, 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 the last request is under the threshold size of 256KB, the last request will be routed to ISA-L software.

Use Powershell Get-Counter command during compression:

```
Get-Counter -Counter "\Intel QuickAssist Accelerator Compression(*)\Compression Bytes In per Interval" `
-SampleInterval 5 -MaxSamples 1
```

Observe that the Compression Bytes is exactly 105,765-Bytes less than the Calgary corpus, or the exact size of the last request size.



```
PS C:\> Get-Counter -Counter $counter -SampleInterval 5 -MaxSamples 1
Timestamp           CounterSamples
-----
1/7/2021 12:04:43 PM \\af11-12-wp2-lbg\intel quickassist accelerator compression(_total)\compression bytes in per
interval :
3145728
```

5.1.2.2 Parcomp Software Fallback Example

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

Warning: Intel® does not recommend intentionally creating hardware errors.

Compressing 'calgary' file into 'calgary_compressed' with DEFLATE_4B, Level 1, 64KB Chunk Size with software 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 cryptography sample application that provides 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(R) QuickAssist Technology\Crypto\Samples\bin

For a list of all the parameters 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 heavily overloading via hundreds of threads per Intel® QAT device, there may be a reduction in observed performance.

5.2.2 Cngtest General Examples

Example running RSA, key size 2048, OAEP padding, in user mode encrypt only:

```
cngtest.exe -provider=qa -algo=rsa -keylength=2048 -padding=oaep -encrypt
```

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin> .\cngtest.exe -provider=qa -algo=rsa -keylength=2048 -padding=oaep -encrypt
----- Platform: Lewisburg
----- Number of Devices: 3
- Max number of threads: 150

Running in user space...
[RunTest] Running with 2 thread(s).
    Time [ns]           : 1861649000
    Number of iterations : 100000
    RSA Encrypt Ops/s    : 53715.82
    CPU core utilization percentage : 0%
    CPU overall utilization percentage : 2%

PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin>

```

Example running ECDH nistP521 curve, in user mode, all operations:

```
cngtest.exe -provider=qa -algo=ecdh -ecccurve=nistP521
```

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin> .\cngtest.exe -provider=qa -algo=ecdh -eccurve=nistP521

----- Platform: Lewisburg
----- Number of Devices: 3
- Max number of threads: 150

Running in user space...
[RunTest] Running with 2 thread(s).
Time [ns] : 85890329900
Number of iterations : 100000
ECDH Stage 1 Ops/s : 1164.28
CPU core utilization percentage : 0%
CPU overall utilization percentage : 3%
Time [ns] : 95565971800
Number of iterations : 100000
ECDH Stage 2 Ops/s : 1046.40
CPU core utilization percentage : 0%
CPU overall utilization percentage : 3%

PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin>

```

Example running ECDSA nistP256 curve, verify operations in user mode:

```
cngtest.exe -provider=qa -algo=ecdsa -eccurve=nistP256 -verify
```

```

Administrator: PowerShell 7 (x64)
PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin> .\cngtest.exe -provider=qa -algo=ecdsa -eccurve=nistP256 -verify

----- Platform: Lewisburg
----- Number of Devices: 3
- Max number of threads: 150

Running in user space...
[RunTest] Running with 2 thread(s).
Time [ns] : 9482750600
Number of iterations : 100000
ECDH Stage 1 Ops/s : 10545.46
CPU core utilization percentage : 0%
CPU overall utilization percentage : 3%
Time [ns] : 11353037200
Number of iterations : 100000
ECDH Stage 2 Ops/s : 8808.22
CPU core utilization percentage : 0%
CPU overall utilization percentage : 3%

PS C:\Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin>

```

5.2.3 Cngtest Fallback

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

Warning: Intel® does not recommend intentionally creating hardware errors.

QATZIP API GUIDE

6.1 QATzip Limitations

The QATzip API is limited to output files of approximately 1.3 GiB. The recommendation for compression and de-compression of large files is to break them up into smaller requests and manage them individually.

The following list of functions are currently not implemented for this release of the QATzip API for Windows* with no plans to implement them:

- qzMemFindAddr
- qzCompressCrc
- qzCompressCrcExt
- qzCompressStream
- qzDecompressCrc64
- qzDecompressCrc64Ext
- qzDecompressStream
- qzEndStream

The following list of functions are currently not implemented for this release of the QATzip API for Windows*:

- qzGetDefaultsDeflate
- qzGetDefaultsLZ4
- qzGetDefaultsLZ4S
- qzSetDefaultsDeflate
- qzSetDefaultsLZ4
- qzSetDefaultsLZ4S
- qzSetupSessionDeflate
- qzSetupSessionLZ4
- qzSetupSessionLZ4S

6.2 Using QATzip Without QAT Hardware

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 systems without Intel® QAT for compatibility or it can be used on systems with Intel® QAT for performance.

The requirement is to initialize qzInit with sw_backup value of 1 or 2. Then for qzSetupSession, set the QzSessionParams_T member *sw_backup* to 1 or 2 and have Intel® ISA-L DLL available in a system path or location defined by the registry key as defined in Intel® ISA-L Location.

Starting with QAT Windows* Release 1.8, it is possible to force the use of a supported software even if the Intel® QAT hardware is available. This requirement is to set the QzSessionParams_T member *sw_backup* to 2 and have the respective software solution available (e.g., Intel® ISA-L). Setting *sw_backup* to 2 in previous Windows QAT driver releases will fail.

6.3 QATzip API Function Overview

This section gives a brief overview of the commonly used QATzip API functions.

For the purposes of Windows* software development, it is recommended to use the qatzip.h, qatzip.dll, and libqatzip.lib that is contained in the Windows* QAT driver package instead of downloading the header file from the QATzip GitHub repository.

6.3.1 Windows* and Linux* QATzip Differences

The QATzip API for Windows* and Linux* are closely aligned, however there are some notable differences as described (but not limited to) the below:

- Windows* QATzip API does not support QZ_DEFLATE_RAW data format.
- Linux* QATzip API does not support QZ_DEFLATE_4B data format.
- Due to Windows* and Linux* not sharing the same data formats, qzGetDefaults behavior has a different data format for Windows* and Linux*.
- Linux* QATzip does not support in-flight software fallback.
- Windows* QATzip relies on Intel® ISA-L for software fallback for Deflate compression and decompression.

6.3.2 qzInit/qzClose Overview

This function initializes the QAT hardware if it is available. The qzClose function closes the initialization.

A brief overview of the sw_backup parameter:

- sw_backup - If set to 1 or 2, qzInit will make sure a software provider (Intel® ISA-L) is available. If no software provider is available, then the return code will be non QZ_OK. This sw_backup field has no relevance in determining software fallback in a compression or decompression operation.

Important: It is up to the application developer to properly use qzTeardownSession and qzClose to free QAT sessions before application termination. Furthermore, do not teardown or close a QAT session while there is an in-flight operation.

6.3.3 qzSessionSetup/qzTeardownSession Overview

This function establishes a QAT session. A QAT session is an association of the hardware used in qzInit and the parameters (QzSessionParams_T). The qzTeardownSession function is used to close the session.

Note: To change the session parameters, the qzTeardownSession must be called first. Afterwards, qzSetupSession can be called with a changed QzSessionParams_T struct. If a qzTeardownSession has not been called before another qzSetupSession, a QZ_DUPLICATE will be returned.

Important: Although it is possible to use qzCompress/qzDecompress without qzInit and qzSetupSession, for Windows* QATzip development, Intel® recommends not doing so.

A brief overview of the member properties of the QzSessionParams_T struct:

- 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 *DEFLATE Data Formats* for more information 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 increases 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 largest 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 chunk size is 64KB, then the max_forks value only needs to be 4 to attain maximum performance.

- sw_backup - Enable, disable, or force software fallback.
 - Value 0 - Disable software path
 - Value 1 - Enables software path if hardware is unavailable
 - Value 2 - Forces software path even if hardware is available

Note: 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 of the request size at which the request will be sent to software for compression/decompression. 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 disabled. The default value is 1 KiB. If the `hw_buff_sz` is smaller than this value, QAT hardware will never be used.

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

6.3.4 qzCompress/qzDecompress Overview

The QATzip API functions `qzCompress` and `qzDecompress` call the `qzCompressExt` and `qzDecompressExt` functions, respectively, except with a NULL `ext_rc`. The effect is ignoring the extended return status.

Prior to Release 1.6, these were the only functions available for QATzip compression and decompression.

Important: It is important to make sure the source/destination buffer and length fields are correct. There may be unexpected behavior if there is a mismatch in either the source/destination buffer or length parameters.

6.3.5 qzCompressExt/qzDecompressExt Overview

Added in Release 1.6.

The QATzip API functions `qzCompressExt/qzDecompressExt`'s purpose is returning an extended return code. An example usage of the extended return code is to determine if the compression or decompression operation was done in software or hardware.

Important: It is important to make sure the source/destination buffer and length fields are correct. There may be unexpected behavior if there is a mismatch in either the sources/destination buffer or length parameters.

6.3.5.1 QATzip Extended Return Codes

The `ext_rc` (extended return code) can be read as follows.

Table 6.1:: QATzip Extended Return Codes

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

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.

Important: 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.

6.3.5.2 qzCompressCrc64Ext/qzDecompressCrc64Ext

This calls qzCompressExt/qzDecompressExt and also returns a 64-bit ECMA-182 CRC64 on the source file for qzCompressCrc64. For qzDecompressCrc64, this also returns a 64-bit ECMA-182 CRC64 on the destination file.

In addition, it returns an extended return code. See [QATzip Extended Return Codes](#) for more information.

6.3.6 qzCompressCrc64/qzDecompressCrc64

Added in Release 2.0

This calls qzCompress/qzDecompress and also returns a 64-bit ECMA-182 CRC64 on the source file for qzCompressCrc64. For qzDecompressCrc64, this also returns a 64-bit ECMA-182 CRC64 on the destination file.

The qzCompressCrc64Ext/qzDecompressCrc64Ext returns an extended return code in addition to the 64-bit ECMA-182 CRC64. See [QATzip Extended Return Codes](#) for more information.

6.3.6.1 qzGetSessionCrc64Config

This will return the default Crc64Config_T struct, given an existing QAT session. Use the API qzSetSessionCrc64Config to set the Crc64Config_T struct.

Note: ISA-L's CRC64 implementation reflects the initial_value and xor_out compared to the QATzip implementation.

6.3.7 getQatVersionIndex Overview

Starting with QAT Windows* Release 1.8, it is now possible to get versioning information from QATzip API for the following components:

- icp_qat - The Windows base driver version. This **does not** match the icp_qat.sys file version. It is an internal kernel driver version shared with the QAT Linux driver.
- ISA-L - The isa-l.dll file version.
- QATzip - The QATzip version, which is based on the Windows package version.

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 [DEFLATE Data Formats](#) for more information regarding the Deflate data formats.

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:

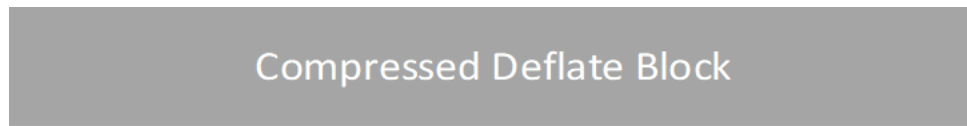


Figure 6.1:: QATzip frame diagram for DEFLATE_RAW

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 had 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:



Figure 6.2:: QATzip frame diagram for DEFLATE_4B

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.

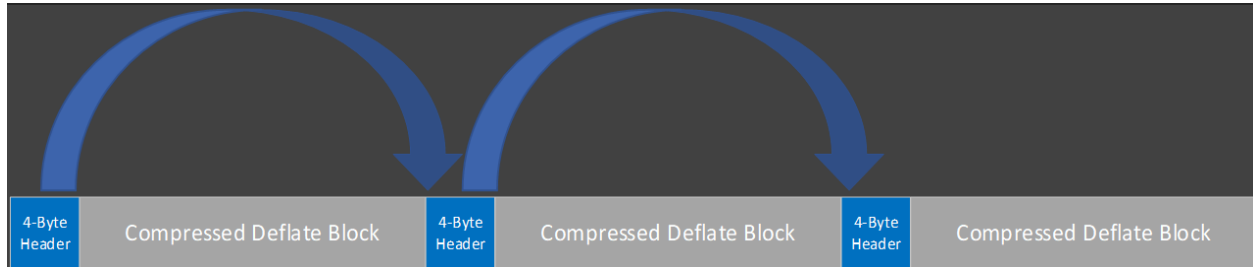


Figure 6.3:: QATzip frame diagram depicting length calculations for DEFLATE_4B

Important: 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. This will result in the data differing from its original compressed form.

6.5.3 DEFLATE_GZIP

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



Figure 6.4:: QATzip frame diagram for DEFLATE_GZIP

A file that is compressed using the DEFLATE_GZIP data format is Gzip compliant and can be decompressed with any standard software Gzip solutions. By 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

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:

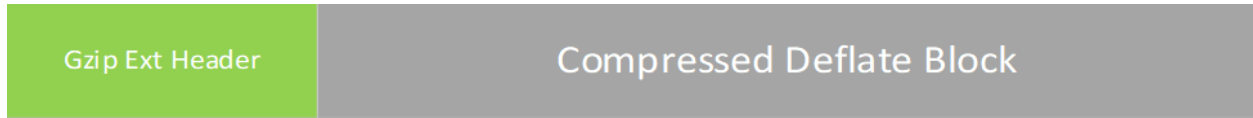


Figure 6.5:: QATzip frame diagram for DEFLATE_GZIP_EXT

The resultant Gzip header using this data format will be slightly larger than 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 [DEFLATE_4B \(Windows* Performance\)](#) 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 standard software Gzip solutions.

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 current Intel® recommended usage model is to expose QAT devices into Virtual Machines using SR-IOV.

Each QAT device supports up to 16 VFs, with Windows* enumeration starting at 0. Multiple VFs 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-, second-, third, and fourth-generation Xeon Scalable platforms. The minimum Windows* version required is Windows* Server 2019 for the host with recent cumulative updates.

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

Important: The QAT VF device must be either removed from the Guest or the Guest must be shut down to bring down (or uninstall) the QAT PF device.

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

7.1.1 PF/VF Software Compatibility

Although it is possible to mix the Windows* QAT PF and VF driver version, Intel® recommends using the same version for the PF and the VF or as close as possible.

7.1.1.1 PF/VF Software Compatibility QAT HW 2.0

The below charts indicate Windows* QAT PF with tested QAT VF drivers for Intel® QAT HW 2.0.

Table 7.1:: Windows* QAT PF with Tested Windows* QAT VF Drivers

Windows* QAT VF Driver	Windows* QAT PF Driver
	W.2.0.1
W.2.0.1	Yes

Table 7.2:: Windows* QAT PF with Tested Linux* QAT VF Drivers

Linux* QAT VF Driver	Windows* QAT PF Driver
	W.2.0.1
L.1.0.0-0011	Yes

7.1.1.2 PF/VF Software Compatibility QAT HW 1.7

The below charts indicate Windows* QAT PF with tested QAT VF drivers for Intel® QAT HW 1.7.

Table 7.3:: Windows* QAT PF with Tested Windows* QAT VF Drivers

Windows* QAT VF Driver	Windows* QAT PF Driver		
	W.1.9.0	W.1.8.0	W.1.7.0
W.1.9.0	Yes	Yes ^{2,3}	No
W.1.8.0	Yes ³	Yes	Yes ³
W.1.7.0	Yes ^{2,3}	Yes ³	Yes
W.1.6.0	Yes ^{1,3}	Yes ^{1,3}	Yes ^{1,3}

Table 7.4:: Windows* QAT PF with Tested Linux* QAT VF Drivers

Linux* QAT VF Driver	Windows* QAT PF Driver		
	W.1.9.0	W.1.8.0	W.1.7.0
L4.19	Yes	Yes ^{2,3}	No
L4.18	Yes ³	Yes ³	No
L4.17	Yes ³	Yes	Yes ³
L4.16	Yes ³	Yes ³	Yes
L4.15	Yes ³	Yes ^{2,3}	Yes ^{2,3}
L4.14	Yes ³	Yes ³	Yes ³
L4.13	Yes ³	Yes ³	Yes ³
L4.12	No	No	Yes ^{1,3}

1 - No Servicing Support

2 - Servicing Support Untested

3 - Limited Validation

Note: While there may be other Windows* and Linux* QAT PF/VF combinations that may work, they have not been validated by Intel®.

7.1.2 QAT VF Device Scaling

To get optimal performance for the Guest, at least one QAT VF device must be attached to the Guest per QAT PF device.

Increasing the number of VFs from the same QAT PF device to the Guest will have greatly diminishing returns in terms of performance.

7.1.3 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 cryptography 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.

7.1.4 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. Starting with Release 1.7, the Windows* VF driver also supports SR-IOV Software Fallback for QATzip compression 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 software fallback requires the QAT VF devices to be removed before doing the software fallback event.

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.

Warning: 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 2022 and newer.

7.1.4.1 SR-IOV Software Fallback Limitations

There are currently some limitations to QAT SR-IOV software fallback.

- The Linux* QAT PF driver does not support SR-IOV software fallback. Therefore, SR-IOV software fallback will not work with a Linux* based hypervisor.
- The ESX* QAT PF driver does not support SR-IOV software fallback. Therefore, SR-IOV software fallback will not work with an ESX* based hypervisor.
- For Linux* Guests, a Windows* QAT PF Release 1.5 and newer is required with Linux* QAT VF Release L4.13 and newer.
- For Windows* Guests, a Windows* QAT PF Release 1.7 and newer is required with Windows* QAT VF Release 1.7 and newer.

- Linux* Guest SR-IOV software fallback is limited to QAT Engine applications.
- The number of QAT VFs present and configured when the QAT Engine application starts is the maximum number that can be used when adding QAT VFs.
- There are limitations when executing multiple concurrent devices add/remove operations on the QAT VFs.

7.2 Enabling SR-IOV on QAT Devices

If the Windows* QAT driver was installed in Hyper-V Mode, all Intel® QAT devices will 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. Please see [SR-IOV Enable](#) for more information on the *EnableSriov* registry key.

An example using PowerShell to enable SR-IOV on a given QAT device:

```
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). A reboot of the system is recommended, but not required.

To confirm SR-IOV is active on the QAT device, use the following PowerShell cmdlet:

```
Get-VMHostAssignableDevice
```

In the example below, there are three QAT devices (VEN_8086&DEV_37C8) that have SR-IOV enabled. Take note of the InstanceId, as value will be used when assigning the QAT VFs.

```
Administrator: c:\windows\system32\cmd.exe
PS C:\> Get-VMHostAssignableDevice

InstanceId      : PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&24A7469D&0&00000000
LocationPath    : PCIROOT(3A)#PCI(0000)#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\6&44B451A&0&00100000
LocationPath    : PCIROOT(D7)#PCI(0000)#PCI(0000)#PCI(0200)#PCI(0000)
CimSession      : CimSession: .
ComputerName    : AF11-12-WP2-LBG
IsDeleted       : False

InstanceId      : PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&86D7F0F&0&00080000
LocationPath    : PCIROOT(3A)#PCI(0000)#PCI(0000)#PCI(0100)#PCI(0000)
CimSession      : CimSession: .
ComputerName    : AF11-12-WP2-LBG
IsDeleted       : False
```

Figure 7.1:: Getting QAT assignable devices via PowerShell

Important: Once SR-IOV has been enabled on a QAT device, that QAT device will no longer be able to do hardware accelerated compression or cryptography services on the Host OS.

7.3 Guest Creation with QAT Virtual Functions

The following steps will walk through a Virtual Machine creation within Hyper-V as well as assigning the QAT VFs.

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

```
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

```
Get-VM -Name "MyVM" | Set-VM -AutomaticStopAction ShutDown
```

4. Add QAT VF devices. In this example, three QAT VFs are added, one from each InstanceId noted in [Enabling SR-IOV on QAT Devices](#):

```
Get-VM -Name "MyVM" |  
  Add-VMAssignableDevice -InstancePath `  
    "PCI\VEN_8086&DEV_37C8&SUBSYS_35CF8086&REV_04\6&24A7469D&0&00000000" `  
  -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 retrieving their status.

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

Note: These cmdlets are for demonstrational purposes only.

- Get all active VFs from a VM with name "MyVM":

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

- Get all active VFs 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"
  }
```

- Remove all QAT VFs (matching DeviceId) from all VM's

```
Get-VM | Get-VMAssignableDevice |
  Where-Object { $_.InstanceId -match "VEN_8086&DEV_37C8"} |
  Remove-VMAssignableDevice
```

- Remove all QAT VFs (matching DeviceId) from a VM with name "MyVM":

```
Get-VM -Name "MyVM" | Get-VMAssignableDevice |
  Where-Object { $_.InstanceId -match "VEN_8086&DEV_37C8"} |
  Remove-VMAssignableDevice
```

- Remove all QAT VFs 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
```

- Add QAT VF #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, follow the same process as installing the QAT drivers on bare-metal or the QAT Windows* PF driver.

See [QAT Driver Install/Uninstall](#) for more information.

Important: For best results, please scan the PCI bus (e.g., via Device Manager) before driver installation to make sure VF devices have been identified by the Windows* Guest.

7.5 QAT Linux* VF Driver Installation

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

Note: More information about the QAT Linux* driver could be found on their respective websites detailed in [Supported Software Versions](#).

Assuming the Linux* QAT driver tarball is located in the home folder (\${HOME}):

1. Install the pre-requisite packages (example for Ubuntu* 18.04 LTS given):

```
sudo 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
```

1. Unpack the QAT Linux* driver tarball:

```
tar -zxvf <QatTarball>
```

Where QatTarball is the file name of the QAT Linux* driver tarball.

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

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

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

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

4. Build and install the driver:

```
sudo make install
```

5. Verify the QAT VFs are up and running:

```
adf_ctl status
```

7.6 Linux* VF Sample Code

The QAT Linux* driver package ships with a sample code application 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 cryptography tests on the driver:

```
cpa_sample_code sign0flife=1
```

7.7 SR-IOV Software Fallback with QAT Engine

An example application that has SR-IOV software 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 software fallback with QAT Engine.

The General Requirements:

- Root and/or Administrator privileges may be required.
- Windows* Host is Windows* Server 2022 with Hyper-V role enabled.
- QAT Windows* PF driver, Release 1.5 or newer in SR-IOV mode.
- The Linux* Guest is Ubuntu* 18.04 LTS with Kernel 4.15.
- QAT Linux* VF driver is Release L4.13 or higher and has been installed as per [QAT Linux* VF Driver Installation](#).
- Async Nginx*
- QAT Engine*
- OpenSSL*

Please refer to [Supported Software Versions](#) on where to get the respective project's source code.

7.7.1 Async Nginx* with QAT Engine Setup

7.7.1.1 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

7.7.1.2 Environmental Variables

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

```
export QAT_SRC=${HOME}/QAT
export OPENSSSL_SRC=${HOME}/openssl
export OPENSSSL_INS=${HOME}/openssl-ins
export NGINX_INS=${HOME}/nginx-ins
export NGINX_SRC=${HOME}/nginx-src
export QATENGINE_SRC=${HOME}/QAT_Engine
export LD_LIBRARY_PATH=$OPENSSSL_INS/lib
export LIBRPATH=$OPENSSSL_INS
export PERL5LIB=$PERL5LIB:$OPENSSSL_SRC
```

7.7.1.3 OpenSSL*/QAT Engine/Async Nginx* Installation

Refer to *Supported Software Versions* for more information on the respective projects and where to download the software packages.

1. Build OpenSSL*:

```
cd $OPENSSSL_SRC
./config --prefix=$OPENSSSL_INS -Wl,-rpath,\${LIBRPATH}
make
make install
```

2. Build QAT-Engine:

```
cd $QATENGINE_SRC
./autogen.sh
./configure \
  | --with-qat_hw_dir=$QAT_SRC \
  | --with-openssl_install_dir=$OPENSSSL_INS \
  | --enable-qat_debug \
  | --with-cc-opt="-DQAT_TESTS_LOG"
make
make install
```

3. 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=$NGINX_SRC/modules/nginx_qat_module \
  | --with-cc-opt="-DNGX_SECURE_MEM -DNGX_INTEL_SDL -I$OPENSSSL_INS/include -Wnoerror=deprecated-  
↪declarations" \
  | --with-ld-opt="-Wl,-rpath=$OPENSSSL_INS/lib -L$OPENSSSL_INS/lib"
make
make install
```

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

7.7.1.4 Async Nginx* Configuration

1. 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;
    }
}

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;

```

(continues on next page)

(continued from previous page)

```
}
}
```

2. Create and save TestServer.cert.pem and TestServer.key.pem (the web server certificate used for authentication):

```
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
```

7.7.2 Testing SR-IOV Software Fallback

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

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

```
#####
# User Process Instance Section
#####
[SHIM]
NumberCyInstances = 2
NumberDcInstances = 2
NumProcesses = 1
LimitDevAccess = 0
```

2. Restart the QAT VFs for the configuration to be active:

```
adf_ctl restart
```

3. In the Linux* Guest, start the Nginx* service:

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

4. In the Linux* Guest, 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
```

5. On the Host Partition, use PowerShell to hot remove all QAT VF devices, assuming the VM name is “MyVM”:

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

6. 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 *Guest Creation with QAT Virtual Functions* and VF 0.

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

7.7.3 Verifying SR-IOV Software Fallback

Check QAT Engine logs for the software fallback event. The log file is 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 VFs 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 this example, there are two crypto instances per VF (NumberCyInstances = 2). Therefore, 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
```

SUPPORT

Trouble tickets can be submitted to: [Resource & Design Center for Development with Intel](#)

For assistance on account setup contact your Intel® representative.