Intel® QuickAssist Technology for Windows*

API Guide

Hardware Version 2.0

Production Release

December 2024

Document Number: 824223

CONTENTS

Intel® QuickAssist Technology for Windows* - API Guide

QAT 2.x Package Version: W.2.4.0-0010

Table 1:: Revision History

Document Number	Revision Number	Description	Revision Date
824223	002	Added intial QATzip API information	December 2024
824223	001	Initial Release	July 2024

CHAPTER

ONE

INTRODUCTION

The Intel® QuickAssist Technology (QAT) accelerator is a high-speed acceleration device that is built into certain Intel® Xeon® processors. The QuickAssist Technology accelerators can be used to offload and accelerate compression, decompression, encryption, and decryption operations in a highly efficient manner.

Each processor in a socket may contain zero, one, two, or four Intel QAT devices, based on the processor SKU. Every QAT device presents itself to the Operating System as a PCI-Express endpoint. While each QAT device end-point functions as an individual device, the software drivers and services can use effective load-balancing strategies to extract the maximum performance from all QAT device(s) on the system.

In this software release, each QAT device supports the following services:

- Compression/Decompression services for various input buffer sizes and at different compression levels using the DEFLATE, LZ4, and LZ4S algorithms.
- Encryption/Decryption services using the Rivest-Shamir-Adleman (RSA), Eliptic Curve Diffie Hellman (ECDH), and Eliptic Curve Digital Signature Algorithm.

1.1 Supported Hardware Platforms

This document applies to the following platforms:

- Intel® Xeon® 4th- and 5th-Generation Scalable Processor with Intel® QAT Gen4.
- Intel® Xeon® 4th- and 5th-Generation Scalable Processor with Intel® QAT Gen4m.
- Intel® Xeon® 6 Processor with Intel® QAT Gen4.

1.2 What's New

- Added initial release of QATzip APIs.

1.3 Terminology

ADF
Acceleration Driver Framework.

AEAD
Authenticated Encryption With Associated Data.

AES
Advanced Encryption Standard.

API
Application Programming Interface.

ASIC
Application Specific Integrated Circuit.

ASYM
Asymmetric Cryptography Service.

BDF
Bus Device Function.

BIOS
Basic Input/Output System.

BSD
Berkeley Software Distribution.

CBC
Cipher Block Chaining mode.

CCM
Counter with CBC-MAC mode.

CIR
Committed Information Rate.

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
ASYM and SYM Cryptography Services.

DC
Data Compression Service.

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 decmopression.

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.

MAC
Message Authentication Code.

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.

PIR
Peak Information Rate.

PF
Physical Function.

PKE
Public Key Encryption.

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

QP
Queue-Pair.

RAS
Reliability, Availability, Serviceability.

RSA
Rivest-Shamir-Adleman.

SAL
Service Access Layer.

SGL
Scatter-Gather List.

SHA
Secure Hash Algorithm.

SLA
Service Level Agreements.

sIOV
Intel® Scalable I/O Virtualization

SKU
Stock Keeping Unit.

SoC
System-on-a-Chip.

SR-IOV
Single-Root Input/Output Virtualization.

SSL
Secure Sockets Layer.

SYM
Symmetric Crypto Service.

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

CHAPTER

TWO

QATZIP

The QATzip API is a Windows* solution to use the QAT accelerator hardware compression and decompression capabilities.

The QATzip API allows for the following features:

- Deflate compression and decompression with 4B header, Gzip, and Gzip Extended header.
- LZ4 compression and decompression.
- LZ4 compression with metadata using XXHash32 or a programmable CRC32/64.
- LZ4S compression that can be post-processed into Zstd.

2.1 QATzip Basic Software Flow

The following is a basic example of the QATzip flow for compression and/or decompression:

- Initialize a QAT session:
  - qzInit
- Set the desired compression parameters (optional):
  - qzGetDefaults (gets default session parameters struct)
  - Change the session parameters struct
  - qzSetDefaults (sets session parameter struct)
- Initialize a QATzip session:
  - qzSetupSession
- Allocate the buffers:
  - qzMalloc (or other preferred memory allocation method)
- Use compress or decompress variations:
  - qzCompress/qzDecompress
- Close the session when done:
  - qzTeardownSession
  - qzClose

2.1.1 QATzip API Scope & Limitations

The QATzip APIs are callable from user mode and kernel mode entities.

The QATzip API is limited to buffer sizes of 1.0 GiB. The recommendation for compression and decompression of large files is to break them up into smaller requests and manage them individually.

In the provided qatzip.h (header) file, not all the APIs have been implemented. Additionally, there may be slight implementation differences between the Windows* and Linux* QATzip.

2.2 QATzip Version

The *QATZIP_API_VERSION* may be used to build QATzip applications that require a minimum set of QATzip features.

Table 2.1:: Revision History

QATZIP_API_VERSION	Description
20500	Added Windows* QATzip implementation support for the metadata compress with programmable CRC32/64.

2.3 Using QATzip Without QAT Hardware

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.

CHAPTER

THREE

QATZIP API REFERENCE

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

For the purposes of Windows* software development, it is recommended to only 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.

3.1 QATzip General Functions

3.1.1 QATzip Versioning

The qzGetSoftwareComponent family allows for the retrieval of the components associated with the QATzip library.

The following components are supported for version in the Windows* QATzip implementation:

- QZ_COMPONENT_KERNEL_DRIVER - The internal kernel version used (this is not the same as the driver file version).
- QZ_COMPONENT_QATZIP_API - The Windows* QATzip version.
- QZ_COMPONENT_SOFTWARE_PROVIDER - The isa-l.dll file version.

3.1.1.1 qzGetSoftwareComponentCount

This function requests the number of software components associated with the QATzip library.

Syntax:

```
int
qzGetSoftwareComponentCount(
    unsigned int *num_elem
);
```

Parameters:

[in, out] uint32_t *num_elem

Pointer to a 32-bit unsigned integer that will populate how many software components are associated with QATzip.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully.

continues on next page

Table 3.1 – continued from previous page

Return Code	Description
QZ_FAIL	The function did not succeed.
QZ_PARAMS	One or more parameters are invalid.
QZ_NO_SW_AVAIL	The function did not find a software provider.
QZ_NO_HW	The function did not find the QAT kernel driver.
QZ_NOSW_NO_HW	The function did not find the QAT kernel driver of the software provider.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

3.1.1.2 qzGetSoftwareComponentVersionList

This function requests the versions of the components associated with the QATzip library.

Syntax:

```
int
qzGetSoftwareComponentVersionList(
    QzSoftwareVersionInfo_T *api_info,
    unsigned int *num_elem
);
```

Parameters:

[in, out] QzSoftwareVersionInfo_T *api_info

Pointer to the *QzSoftwareVersionInfo_T* struct to populate with version information.

[in, out] uint32_t *num_elem

Pointer to a 32-bit unsigned integer that will populate how many software components are associated with QATzip.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully.
QZ_FAIL	The function did not succeed.
QZ_PARAMS	One or more parameters are invalid.
QZ_NO_SW_AVAIL	The function did not find a software provider.
QZ_NO_HW	The function did not find the QAT kernel driver.
QZ_NOSW_NO_HW	The function did not find the QAT kernel driver of the software provider.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

3.2 QATzip Initialization Functions

3.2.1 qzInit

This function initializes the QAT hardware and required resources including, but not limited to:

- QAT hardware
- Contiguous memory for buffers

Syntax:

```
int
qzInit(
   QzSession_T *sess,
   unsigned char sw_backup
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] unsigned char sw_backup

The sw_backup behavior to use.

- 0 - This will not look for a software backup solution at initialization time.
- 1 - This will look for a software backup solution at initialization time, if none is found, it will not return *QZ_OK*.
- 2 - This will look for a software backup solution at initialization time, if none is found, it will not return *QZ_OK*. Additionally, the session will be software only, if supported.

Note: Invoking *qzInit* with *sw_backup* as values 0 or 1, you can override and allow for software failover or the software only path by setting the *sw_backup* member in *QzSessionParams_T* or *QzSessionParamsCommon_T* to value 1 or 2, respectively.

However, invoking *qzInit* with *sw_backup* as value 2 will always be software only regardless of the *sw_backup* value in *QzSessionParams_T* or *QzSessionParamsCommon_T*.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and a QAT session has been established.
QZ_DUPLICATE	This process or thread already has a QAT session.
QZ_PARAMS	One or more parameters are invalid.
QZ_NOSW_NO_HW	No QAT HW or SW initialization possible.
QZ_NO_SW_AVAIL	QAT SW backup requested, but no QAT SW available.
QZ_NOSW_LOW_MEM	Failed to allocate internal structures.

Remarks:

This function is synchronous.

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 are in-flight operations.

3.2.2 qzSetupSession Family

This function family establishes a QATzip session. Before this function is called, the hardware (and/or software) must have been successfully started via qzInit.

There are several variations of this function for the respective compression algorithms:

- qzSetupSession
- qzSetupSessionDeflate
- qzSetupSessionLZ4
- qzSetupSessionLZ4S

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and a hardware or software based compression session has been created.
QZ_DUPLICATE	The *sess* includes an existing hardware or software session.
QZ_PARAMS	One or more parameters are invalid.
QZ_NOSW_NO_HW	QATzip session has not been initialized.

Remarks:

This function is synchronous.

Note that in-flight changing of compression parameters is not supported. If a *qzTeardownSession* has not been called before another *qzSetupSession*, a QZ_DUPLICATE will be returned.

If a change in compression parameters is required, the flow should be:

- Wait for in-flight compression operations to finish.
- Teardown the session via *qzTeardownSession*.
- Change the *QzSessionParams_T* struct (or respective variant) to desired value.
- Start the session via *qzSetupSession* (or respective variant). The new desired value shall take effect.

Important: For certain file formats, it is up to the user to store the compression parameters used to compress. Decompressing data using parameters different from compression may result in unreadable data.

Important: Although it is currently possible to use *qzCompress* or *qzDecompress* variants without first calling *qzInit* and *qzSetupSession*, for Windows* QATzip development, this is not the recommended flow.

3.2.2.1 qzSetupSession

This function is the legacy setup session function that perform Deflate (default) or LZ4.

Syntax:

```
int
qzSetupSession(
   QzSession_T *sess,
   QzSessionParams_T *params
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[out] QzSessionParams_T *params

Compression parameters contained with the *QzSessionParams_T* struct.

3.2.2.2 qzSetupSessionDeflate

This function is the deflate specific setup session function.

Syntax:

```
int
qzSetupSessionDeflate(
   QzSession_T *sess,
   QzSessionParamsDeflate_T *params
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[out] QzSessionParamsDeflate_T *params

Compression parameters contained with the *QzSessionParamsDeflate_T* struct.

3.2.2.3 qzSetupSessionLZ4

This function is the LZ4 specific setup session function.

```
int
qzSetupSessionLZ4(
   QzSession_T *sess,
   QzSessionParamsLZ4_T *params
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[out] QzSessionParamsLZ4_T *params

Compression parameters contained with the *QzSessionParamsLZ4_T* struct.

3.2.2.4 qzSetupSessionLZ4S

This function is the LZ4S specific setup session function.

Syntax:

```
int
qzSetupSessionLZ4S(
   QzSession_T *sess,
   QzSessionParamsLZ4S_T *params
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[out] QzSessionParamsLZ4S_T *params

Compression parameters contained with the *QzSessionParamsLZ4S_T* struct.

3.2.3 qzTeardownSession

This function is used to uninitialize a QATzip session and deallocates buffers.

If no session has been initialized, no action will take place.

Syntax:

```
int
qzTeardownSession(
   QzSession_T *sess
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and the QATzip session has been uninitialized.
QZ_FAIL	The function did not succeed in uninitializing the QATzip session.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous.

3.2.4 qzClose

This function closes the connection with the QAT hardware and frees the requisite resources.

Syntax:

```
int
qzClose(
   QzSession_T *sess
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and the QAT session has been closed.
QZ_FAIL	The function did not succeed in closing the QAT session.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous.

3.2.5 QATzip CRC Session Functions

This function family gets and sets the programmable CRC32 or CRC64 value of the current QAT session.

Note: The current Windows* QATzip implementation does not support a programmable CRC32 for the *qzCompress* or *qzDecompress* family of functions. A programmable CRC32 is only supported via *qzCompressWithMetadataExt*.

3.2.5.1 qzGetSessionCrc32Config

This function requests the CRC32 configuration of the provided QAT session.

This will populate the *QzCrc32Config_T* with the current CRC32 configuration of the QAT session. This has a dependency of invoking a *qzSetupSession* family first.

Syntax:

```
int
qzGetSessionCrc32Config(
   QzSession_T *sess,
   QzCrc32Config_T *crc32_config
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] QzCrc32Config_T *crc32_config

Pointer to *QzCrc32Config_T* that will return the CRC32 information.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully.
QZ_FAIL	The function did not succeed; session was not setup.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

3.2.5.2 qzGetSessionCrc64Config

This function requests the CRC64 configuration of the provided QAT session.

This will populate the *QzCrc64Config_T* with the current CRC64 configuration of the QAT session. This has a dependency of invoking a *qzSetupSession* family first.

Syntax:

```
int
qzGetSessionCrc64Config(
   QzSession_T *sess,
   QzCrc64Config_T *crc64_config
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] QzCrc64Config_T *crc64_config

Pointer to *QzCrc64Config_T* that will return the CRC64 information.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully.
QZ_FAIL	The function did not succeed; session was not setup.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

3.2.5.3 qzSetSessionCrc32Config

This function sets the CRC32 configuration of the provided QAT session with user defined parameters.

Syntax:

```
int
qzSetSessionCrc32Config(
   QzSession_T *sess,
   QzCrc32Config_T *crc32_config
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] QzCrc32Config_T *crc32_config

Pointer to *QzCrc32Config_T* with user defined values to set to *sess*.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully.
QZ_FAIL	The function did not succeed; session was not setup.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

This function will return non-QZ_OK if invoked after the session has initiated compression or decompression operation. To set the CRC, use *qzTeardownSession*, a *qzSetupSession* variant, then *qzSetSessionCrc32Config*.

3.2.5.4 qzSetSessionCrc64Config

This function sets the CRC64 configuration of the provided QAT session with user defined parameters.

Syntax:

```
int
qzSetSessionCrc64Config(
   QzSession_T *sess,
   QzCrc64Config_T *crc64_config
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] QzCrc64Config_T *crc64_config

Pointer to *QzCrc64Config_T* with user defined values to set to *sess*.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully.
QZ_FAIL	The function did not succeed; session was not setup.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

This function will return non-QZ_OK if invoked after the session has initiated compression or decompression operation. To set the CRC, use *qzTeardownSession*, a *qzSetupSession* variant, then *qzSetSessionCrc64Config*.

3.3 QATzip General Data Types

3.3.1 QzSession_T

This structure contains the pointer to the QAT session and session state.

Syntax:

```
typedef struct QzSession_S
{
   signed long int hw_session_stat;
   int thd_sess_stat;
   void *internal;
   unsigned long total_in;
   unsigned long total_out;
} QzSession_T;
```

Value	Description
hw_session_stat	Opaque value filled during session initialization and use.
thd_sess_stat	The process compression and decompression thread state.
internal	The opaque pointer to the session data.
total_in	Total processed input data this session.
total_out	Total processed output data this session.

3.3.2 QzSoftwareComponentType_T

This enum contains the type of software being described.

Syntax:

```
typedef enum QzSoftwareComponentType_E
{
   QZ_COMPONENT_FIRMWARE = 0,
   QZ_COMPONENT_KERNEL_DRIVER,
   QZ_COMPONENT_USER_DRIVER,
   QZ_COMPONENT_QATZIP_API,
   QZ_COMPONENT_SOFTWARE_PROVIDER
} QzSoftwareComponentType_T;
```

Value	Description
QZ_COMPONENT_FIRMWARE	The firmware version. This is not supported in Windows* QATzip implementation.
QZ_COMPONENT_KERNEL_DRIVER	The kernel driver version.
QZ_COMPONENT_USER_DRIVER	The user driver version. This is not supported in Windows* QATzip implementation.
QZ_COMPONENT_QATZIP_API	The QATzip API version.
QZ_COMPONENT_SOFTWARE_PROV	The QATzip software backup version.

3.3.3 QzSoftwareVersionInfo_T

This structure contains QATzip version data.

Syntax:

```
#define QZ_MAX_STRING_LENGTH 64
typedef struct QzSoftwareVersionInfo_S
{
   QzSoftwareComponentType_T component_type;
   unsigned char component_name[QZ_MAX_STRING_LENGTH];
   unsigned int major_version;
   unsigned int minor_version;
   unsigned int patch_version;
   unsigned int build_number;
   unsigned char reserved[52];
} QzSoftwareVersionInfo_T;
```

Value	Description
QzSoftwareComponentType_T	Opaque value filled during session initialization and use.
component_name	The component name.
major_version	The component major version.
minor_version	The component minor version.
patch_version	The component patch version.
build_version	The component build version.
reserved	Reserved data.

3.4 QATzip Data Compression Functions

3.4.1 qzCompress Family

This function family compresses a buffer if either the hardware and/or software based session is available.

There are several variations of this function for optional CRC64 and extended return code:

- qzCompress
- qzCompressExt
- qzCompressCrc64
- qzCompressCrc64Ext

For functions that support the extended return code (*ext_rc*), the returned value can be read as follows:

Table 3.14:: QATzip Extended Return Codes

Bits	Type	Name	Description
63:5	Bits		Undefined, reserved for future use.
4	Bit	QZ_SW_EXECUTION_BIT	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.

Important: The 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.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and the buffer has been compressed with updated crc and ext_rc, if applicable.
QZ_FAIL	The function did not succeed in compressing the buffer.
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

3.4.1.1 qzCompress

This function is the baseline compression function.

Syntax:

```
int
qzCompress(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   unsigned int last
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be compressed.

[in, out] unsigned int *src_len

Pointer to the length of the source buffer. This will be modified to the number of bytes consumed.

[in] unsigned char *dest

Pointer to the destination buffer for the compressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the compressed buffer when the function returns successfully. This value may be zero if no data is produced.

[in] unsigned int last

Default value is 0, however, this is not used in the Windows* QATzip implementation.

3.4.1.2 qzCompressExt

This function is the compression function with extended return code.

Syntax:

```
int
qzCompressExt(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   unsigned int last,
   uint64_t *ext_rc
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be compressed.

[in, out] unsigned int *src_len

Pointer to the length of the source buffer. This will be modified to the number of bytes consumed.

[in] unsigned char *dest

Pointer to the destination buffer for the compressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the compressed buffer when the function returns successfully. This value may be zero if no data is produced.

[in] unsigned int last

Default value is 0, however, this is not used in the Windows* QATzip implementation.

[in, out] uint64_t *ext_rc

Pointer to the extended return code. If NULL, no extended return code information will be provided (effectively the same as using the non-Ext compression functions).

See *QATzip Extended Return Codes Table* for more details.

3.4.1.3 qzCompressCrc64

This function is the compression function that also returns a programmable CRC64 checksum of the source buffer.

Syntax:

```
int
qzCompressCrc64(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   unsigned int last,
   uint64_t *crc
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be compressed.

[in, out] unsigned int *src_len

Pointer to the length of the source buffer. This will be modified to the number of bytes consumed.

[in] unsigned char *dest

Pointer to the destination buffer for the compressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the compressed buffer when the function returns successfully. This value may be zero if no data is produced.

[in] unsigned int last

Default value is 0, however, this is not used in the Windows* QATzip implementation.

[in, out] uint64_t *crc

Pointer to CRC64 checksum buffer. This will be modified to the CRC64 checksum when the function returns successfully.

The default CRC64 used is ECMA-182.

3.4.1.4 qzCompressCrc64Ext

This function is the compression function that returns a programmable CRC64 checksum of the source buffer and also the extended return code.

Syntax:

```
int
qzCompressCrc64Ext(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   unsigned int last,
   uint64_t *crc,
   uint64_t *ext_rc
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be compressed.

[in, out] unsigned int *src_len

Pointer to the length of the source buffer. This will be modified to the number of bytes consumed.

[in] unsigned char *dest

Pointer to the destination buffer for the compressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the compressed buffer when the function returns successfully. This value may be zero if no data is produced.

[in] unsigned int last

Default value is 0, however, this is not used in the Windows* QATzip implementation.

[in, out] uint64_t *crc

Pointer to CRC64 checksum buffer. This will be modified to the CRC64 checksum when the function returns successfully.

The default CRC64 used is ECMA-182.

[in, out] uint64_t *ext_rc

Pointer to the extended return code. If NULL, no extended return code information will be provided (effectively the same as using the non-Ext compression functions).

See *QATzip Extended Return Codes Table* for more details.

3.4.2 qzCompress2/qzCompressCrc/qzCompressCrcExt

These functions are not implemented in the current Windows* QATzip implementation.

3.4.3 qzCompressWithMetadataExt

This function compresses a buffer and writes metadata for each compressed block into an opaque metadata structure.

The default CRC used for the each block is XXHash32. However, a programmable CRC32 and CRC64 is supported.

Syntax:

```
int
qzCompressWithMetadataExt(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   unsigned int last,
   uint64_t *ext_rc,
   QzMetadataBlob_T metadata,
   uint32_t hw_buff_sz_override,
   uint32_t comp_thrshold
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be compressed.

[in, out] unsigned int *src_len

Pointer to the length of the source buffer. This will be modified to the number of bytes consumed.

[in] unsigned char *dest

Pointer to the destination buffer for the compressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the compressed buffer when the function returns successfully. This value may be zero if no data is produced.

[in] unsigned int last

Default value is 0, however, this is not used in the Windows* QATzip implementation.

[in, out] uint64_t *ext_rc for *qzCompressExt* and *qzCompressCrc64Ext*

Pointer to the extended return code. If NULL, no extended return code information will be provided (effectively the same as using the non-Ext compression functions).

See *QATzip Extended Return Codes Table* for more details.

[in] QzMetadataBlob_T metadata

Opaque metadata structure(s) that contains the metadata for each compressed block.

[in] uint32_t hw_buff_sz_override

This value specifies the data size to be used for each compression operation. It will override the *hw_buff_sz* parameter specified at session creation. If value 0 is specified, then the *hw_buff_sz* parameter specified at session creation will be used.

[in] uint32_t comp_thrshold

This values specifies the compression threshold of a block. If the compressed size is greater than the threshold, this function will copy the compressed data to the destination buffer and set the size of the block in the metadata to the size of the uncompressed block.

Otherwise, the compressed data will be copied to the destination buffer and set the size of the block in the metadata to the size of the compressed block.

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and the buffer has been compressed with ext_rc, if applicable.
QZ_FAIL	The function did not succeed in compressing the buffer.
QZ_PARAMS	One or more parameters are invalid.
QZ_METADATA_OVERFLOW	Unable to populate metadata due to insufficient memory allocated.
QZ_NOT_SUPPORTED	The selected algorithm or data format is not supported for metadata compression.
QZ_NOSW_NO_HW	No kernel driver or software provider could be found.
QZ_NOSW_NO_INS_ATTACH	No instance available.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

Memory for the opaque metadata structure should be allocated based on the *hw_buff_sz* or the *hw_buff_sz_override* that is used in the compression operation.

It is best practice to call *qzInit* and *qzSetupSession* variant (with the desired compression parameters) before using this function.

3.4.4 qzDecompress Family

This function family decompresses a buffer if either the hardware and/or software based session is available.

There are several variations of this function for optional CRC64 and extended return code:

- qzDecompress
- qzDecompressExt
- qzDecompressCrc64
- qzDecompressCrc64Ext

Return Value:

Return Code	Description
QZ_OK	The function executed successfully and the buffer has been decompressed with updated crc and ext_rc, if applicable.
QZ_FAIL	The function did not succeed in decompressing the buffer.

continues on next page

Table 3.17 – continued from previous page

Return Code	Description
QZ_PARAMS	One or more parameters are invalid.

Remarks:

This function is synchronous and shall not be called in an interrupt context.

3.4.4.1 qzDecompress

This function is the decompression function that also returns the extended return code.

Syntax:

```
int
qzDecompress(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be decompressed.

[in] unsigned int *src_len

Pointer to the length of the source buffer.

[in] unsigned char *dest

Pointer to the destination buffer for the decompressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the decompressed buffer when the function returns successfully.

3.4.4.2 qzDecompressExt

This function is the decompression function that also returns the extended return code.

Syntax:

```
int
qzDecompressExt(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
```

(continues on next page)

(continued from previous page)

```
   unsigned int *dest_len,
   uint64_t *ext_rc
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be decompressed.

[in] unsigned int *src_len

Pointer to the length of the source buffer.

[in] unsigned char *dest

Pointer to the destination buffer for the decompressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the decompressed buffer when the function returns successfully.

[in, out] uint64_t *ext_rc

Pointer to the extended return code. If NULL, no extended return code information will be provided (effectively the same as using the non-Ext compression functions).

See *QATzip Extended Return Codes Table* for more details.

3.4.4.3 qzDecompressCrc64

This function is the decompression function that verifies that the programmable CRC64 checksum matches the destination buffer.

Syntax:

```
int
qzDecompressCrc64(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   uint64_t *crc
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be decompressed.

[in] unsigned int *src_len

Pointer to the length of the source buffer.

[in] unsigned char *dest

Pointer to the destination buffer for the decompressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the decompressed buffer when the function returns successfully.

[in, out] uint64_t *crc

Pointer to CRC64 checksum buffer. This will be modified to the CRC64 checksum when the function returns successfully.

The default CRC64 used is ECMA-182.

3.4.4.4 qzDecompressCrc64Ext

This function is the decompression function that verifies that the programmable CRC64 checksum matches the destination buffer and also returns the extended return code.

Syntax:

```
int
qzDecompressCrc64Ext(
   QzSession_T *sess,
   const unsigned char *src,
   unsigned int *src_len,
   unsigned char *dest,
   unsigned int *dest_len,
   uint64_t *crc,
   uint64_t *ext_rc
);
```

Parameters:

[in] QzSession_T *sess

Session handle that is a pointer to opaque instance and session data.

[in] const unsigned char *src

Pointer to the source buffer to be decompressed.

[in] unsigned int *src_len

Pointer to the length of the source buffer.

[in] unsigned char *dest

Pointer to the destination buffer for the decompressed data to be copied.

[in, out] unsigned int *dest_len

Pointer to the length of the destination buffer. This will be modified to the actual length of the decompressed buffer when the function returns successfully.

[in, out] uint64_t *crc

Pointer to CRC64 checksum buffer. This will be modified to the CRC64 checksum when the function returns successfully.

The default CRC64 used is ECMA-182.

[in, out] uint64_t *ext_rc

Pointer to the extended return code. If NULL, no extended return code information will be provided (effectively the same as using the non-Ext compression functions).

See *QATzip Extended Return Codes Table* for more details.

3.4.5 qzDecompress2/qzDecompressCrc/qzDecompressCrc64Ext

These functions are not implemented in the current Windows* QATzip implementation.

3.4.6 qzDecompressWithMetadataExt

This function is not implemented in the current Windows* QATzip implementation.

3.5 QATzip Data Compression Data Types

3.5.1 QzCrc32Config_T

This structure contains data relating to the session's CRC32 settings.

Syntax:

```
typedef struct QzCrc32Config_S
{
   uint32_t polynomial;
   uint32_t initial_value;
   uint32_t reflect_in;
   uint32_t reflect_out;
   uint32_t xor_out;
} QzCrc32Config_T;
```

Value	Description
polynomial	Polynomial used for the CRC32 calculation.
initial_value	Intial CRC32 value. Default value is 0.
reflect_in	Reflect bit order before CRC32 calculation. Default value is 0.
reflect_out	Reflect bit order after CRC32 calculation. Default value is 0.
xor_out	XOR out value for the the CRC32 calculation. Default value is 0.

3.5.2 QzCrc64Config_T

This structure contains data relating to the session's CRC64 settings.

The default CRC64 polynomial used is ECMA-182 Normal.

Syntax:

```
typedef struct QzCrc64Config_S
{
   uint32_t polynomial;
   uint32_t initial_value;
   uint32_t reflect_in;
   uint32_t reflect_out;
   uint32_t xor_out;
} QzCrc64Config_T;
```

Value	Description
polynomial	Polynomial used for the CRC64 calculation. Default value is 0x42F0E1EBA9EA3693.
initial_value	Intial CRC64 value. Default value is 0.
reflect_in	Reflect bit order before CRC64 calculation. Default value is 0.
reflect_out	Reflect bit order after CRC64 calculation. Default value is 0.
xor_out	XOR out value for the the CRC64 calculation. Default value is 0.

3.5.3 QzDataFormat_T

This enum identifies the Deflate data format.

Syntax:

```
typedef struct QzSessionParamsDeflate_S {
   QZ_DEFLATE_4B = 0,
   QZ_DEFLATE_GZIP,
   QZ_DEFLATE_GZIP_EXT,
   QZ_DEFLATE_RAW,
   QZ_FMT_NUM
} QzDataFormat_T;
```

Value	Description
QZ_DEFLATE_4B	Data is raw Deflate with a 4 byte length header.
QZ_DEFLATE_GZIP	Data is Deflate wrapped by Gzip header and footer.
QZ_DEFLATE_GZIP_EXT	Data is Deflate wrapped by Gzip extended header (contains a length field) and footer.
QZ_DEFLATE_RAW	Data is raw Deflate. This is not supported in Windows* QATzip implementation.
QZ_FMT_NUM	This represents the length of the this enum.

3.5.4 QzDirection_T

This enum identifies the direction.

Syntax:

```
typedef enum QzDirection_E {
   QZ_DIR_COMPRESS = 0,
   QZ_DIR_DECOMPRESS,
   QZ_DIR_BOTH
} QzDirection_T;
```

Value	Description
QZ_DIR_COMPRESS	Session will be used for compression.
QZ_DIR_DECOMPRESS	Session will be used for decompression.
QZ_DIR_BOTH	Session will be used for compression and decompression.

3.5.5 QzHuffmanHdr_T

This enum identifies the Huffman header types.

Syntax:

```
typedef enum QzHuffmanHdr_E {
   QZ_DYNAMIC_HDR = 0,
   QZ_STATIC_HDR
} QzHuffmanHdr_T;
```

Value	Description
QZ_DYNAMIC_HDR	Full dynamic Huffman trees used.
QZ_STATIC_HDR	Static Huffman trees used.

3.5.6 QzSessionParams Family

This family of structures is the QATzip session parameters that can be used to initialize a QATzip session.

It consists of algorithm specific structures:

- QzSessionParamsCommon_T
- QzSessionParamsDeflate_T
- QzSessionParamsLZ4_T
- QzSessionParamsLZ4S_T

3.5.6.1 QzSessionParams_T

This structure is the legacy QATzip session parameters that be used for Deflate (default) or LZ4 compression and decompression operations.

For the *sw_backup* member, some algorithms may not have a QATzip based software failover solution. The supported QATzip software backup providers are using Deflate and LZ4 algorithms. For the Deflate algorithm, the Intel® ISA-L DLL must be available on the target system.

Syntax:

```
typedef struct QzSessionParams_S {
   QzHuffmanHdr_T huffman_hdr;
   QzDirection_T direction;
   QzDataFormat_T data_fmt;
   unsigned int comp_lvl;
   unsigned char comp_algorithm;
   unsigned int max_forks;
   unsigned char sw_backup;
```

(continues on next page)

(continued from previous page)

```
   unsigned int hw_buff_sz;
   unsigned int strm_buff_sz;
   unsigned int input_sz_thrshold;
   unsigned int req_cnt_thrshold;
   unsigned int wait_cnt_thrshold;
} QzSessionParams_T;
```

Value	Description
huffman_hdr	Uses *QzHuffmanHdr_T* to configures dynamic or static huffman headers. This is only used for Deflate algorithm. Default value is *QZ_DYNAMIC_HDR*.
direction	Uses *QzDirection_T* to configure the session direction. Default value is *QZ_DIR_BOTH*.
data_fmt	Uses *QzDataFormat_T* to configure the output data format. Default value is *QZ_DEFLATE_4B*.
comp_lvl	The compression level to use. The actual compression level will vary based on QAT hardware. E.g., for QatGen4, actual hardware levels are 1, 6, and 9. With the values inbetween remapped by software. Default value is 1. Range is 1 to 9.
comp_algorithm	The compression algorithm to use. Default value is '8' (*QZ_DEFLATE*). For *QZ_LZ4*, use value '4'.
max_forks	The maximum number of concurrent requests to hardware per QATzip compress or decompress operation. A value of 0 means no forking is permitted. The number of total jobs submitted to QAT hardware per QATzip compression operation is equal to the (input buffer size / hw_buff_size) rounded up to the next largest integer E.g., if the maximum offload size for an app is 256KiB, and the chunk size is 64KiB, then the *max_forks* value only needs to be 4 to achieve best performance. Performance gains can be seen in specific use cases up to 60, though the gains diminishes as *max_forks* increases. In most use cases, greater than 60 results in very little change in performance. Default value is 30. Range is 0 to 100.
sw_backup	The software backup configuration. 0 - No software backup solution will be used. 1 - Software backup will be used if HW fails or threshold. 2 - Software will always be used. Default value is 1.
hw_buff_sz	The buffer size to use. The optimal value may vary with the workload, but the general observation is 32KiB or 64KiB are the general optimal values. Default value is (64 * 1024). Range is 1024 to (1024 * 1024).
strm_buff_sz	The stream buffer size to use. This is not used in Windows* QATzip implementation. Default value is *hw_buff_sz*.
input_sz_thrshold	The threshold of request size at which the request will be sent to software for compression and decompression. This will not work if *sw_backup* is 0. It is up to the user to determine what the best threshold values are, as workloads vary. One must consider a function of acceptable CPU utilization, latency requirements, and desired throughput. Default value is 1024. Range is 128 to 2^31.

continues on next page

Table 3.23 – continued from previous page

Value	Description
wait_cnt_thrshold	The number of calls to wait if a request fails. Default value is 8.

3.5.6.2 QzSessionParamsCommon_T

This structure is the common QATzip session parameters that is used as part of *QzSessionParamsDeflate_S*, *QzSessionParamsLZ4_S*, or *QzSessionParamsLZ4S_S*.

Syntax:

```
typedef struct QzSessionParamsCommon_S {
   QzDirection_T direction;
   unsigned int comp_lvl;
   unsigned char comp_algorithm;
   unsigned int max_forks;
   unsigned char sw_backup;
   unsigned int hw_buff_sz;
   unsigned int strm_buff_sz;
   unsigned int input_sz_thrshold;
   unsigned int req_cnt_thrshold;
   unsigned int wait_cnt_thrshold;
   QzPollingMode_T polling_mode;
   unsigned int is_sensitive_mode;
} QzSessionParamsCommon_T;
```

Value	Description
direction	Uses *QzDirection_T* to configure the session direction. Default value is *QZ_DIR_BOTH*.
comp_lvl	The compression level to use. The actual compression level will vary based on QAT hardware. E.g., for QatGen4, actual hardware levels are 1, 6, and 9. With the values inbetween remapped by software. Default value is 1. Range is 1 to 9.
comp_algorithm	The compression algorithm to use. Default value is '8' (*QZ_DEFLATE*). For *QZ_LZ4*, use value '4'.
max_forks	The maximum number of concurrent requests to hardware per QATzip compress or decompress operation. A value of 0 means no forking is permitted. The number of total jobs submitted to QAT hardware per QATzip compression operation is equal to the (input buffer size / hw_buff_size) rounded up to the next largest integer E.g., if the maximum offload size for an app is 256KiB, and the chunk size is 64KiB, then the *max_forks* value only needs to be 4 to achieve best performance. Performance gains can be seen in specific use cases up to 60, though the gains diminishes as *max_forks* increases. In most use cases, greater than 60 results in very little change in performance. Default value is 30. Range is 0 to 100.
sw_backup	The software backup configuration. Default value is 1. Range is 0 to 2.
hw_buff_sz	The buffer size to use. Default value is (64 * 1024). Range is 1024 to (1024 * 1024).

continues on next page

Table 3.24 – continued from previous page

Value	Description
strm_buff_sz	The stream buffer size to use. This is not used in Windows* QATzip implementation. Default value is *hw_buff_sz*.
input_sz_thrshold	The threshold of request size at which the request will be sent to software for compression and decompression. This will not work if *sw_backup* is 0. It is up to the user to determine what the best threshold values are, as workloads vary. One must consider a function of acceptable CPU utilization, latency requirements, and desired throughput. Default value is 1024. Range is 128 to 2^31.
wait_cnt_thrshold	The number of calls to wait if a request fails. Default value is 8.
polling_mode	The polling mode to use. This is not used in Windows* QATzip implementation.
is_sensitive_mode	Enable or disable sensitive mode. This is not used in Windows* QATzip implementation.

3.5.6.3 QzSessionParamsDeflate_T

This structure is the Deflate algorithm specific QATzip session parameters.

Syntax:

```
typedef struct QzSessionParamsDeflate_S {
   QzSessionParamsCommon_T common_params;
   QzHuffmanHdr_T huffman_hdr;
   QzDataFormat_T data_fmt;
} QzSessionParamsDeflate_T;
```

Value	Description
common_params	Uses *QzSessionParamsCommon_T* to configure the common parameters.
huffman_hdr	Uses *QzHuffmanHdr_T* to configures dynamic or static huffman headers. This is only used for Deflate algorithm. Default value is *QZ_DYNAMIC_HDR*.
data_fmt	Uses *QzDataFormat_T* to configure the output data format. Default value is *QZ_DEFLATE_4B*.

3.5.6.4 QzSessionParamsLZ4_T

This structure is the LZ4 algorithm specific QATzip session parameters.

Syntax:

```
typedef struct QzSessionParamsLZ4_S {
   QzSessionParamsCommon_T common_params;
} QzSessionParamsLZ4_T;
```

Value	Description
common_params	Uses *QzSessionParamsCommon_T* to configure the common parameters.

3.5.6.5 QzSessionParamsLZ4S_T

This structure is the LZ4S algorithm specific QATzip session parameters.

Syntax:

```
typedef struct QzSessionParamsLZ4S_S {
   QzSessionParamsCommon_T common_params;
   qzLZ4SCallbackFn qzCallback;
   void *qzCallback_external;
   unsigned int lz4s_mini_match;
} QzSessionParamsLZ4S_T;
```

Value	Description
common_params	Uses *QzSessionParamsCommon_T* to configure the common parameters.
qzCallback	The post processing callback for Zstd compression.
qzCallback_external	An user provided opaque pointer to be passed into the *qzCallback* during Zstd post processing.
lz4s_mini_match	The LZ4S dictionary mini match. Default value is 3. Range is 3 to 4.

CHAPTER

FOUR

RATE LIMITING

Rate limiting is a solution designed in the QAT accelerator software to enforce Service Level Agreements (SLA). SLA's allocates a specified amount of acceleration capacity for a specified service, such as PKE (ASYM) and data compression (DC) at a queue-pair (QP) granularity.

The Committed Information Rate (CIR) is the the guaranteed rate and the Peak Information Rate (PIR) is the maximum rate allowed for the queue-pair. These values are configured for the DC and ASYM services separately.

The rate limiting solution provides the following features:

- Ability to query rate limiting information for QAT instances in the Guest/Host
- Ability to query rate limiting information for QAT devices in the Host
- Ability to configure rate limiting SLA for services based on QP's

4.1 Rate Limiting Software Flow (Host)

The following is a rate limiting flow from the Host OS's perspective:

- Host Start
- Assign VFs to Guest(s)
- Enable rate limiting on a device using cpaEnableRateLimiting
- Get device level SLA information using:
  - cpaGetDevRlPropertiesHandle
  - cpaRlPropGetNumInterfaces
  - cpaGetDevRlPropSlaCir
  - cpaGetDevRlPropSlaPir
- Get QP handles for managing SLA using:
  - cpaRlGetQpNumHandles
  - cpaRlGetQpHandles
- Get/Set/Delete SLA PIR/CIR on QP using:
  - cpaSetRlSla
  - cpaDeleteRlSla
  - cpaGetRlSla

4.2 Rate Limiting Software Flow (Guest)

The following is a rate limiting flow from the Guest OS's perspective (assuming Host has been properly configured):

- Guest Start
- Discover Instances
- Get SLA information using:
  - cpaGetInstanceRlSlaPir
  - cpaGetInstanceRlSlaCir
- Use the instances

Note: The Guest cannot set SLA's.

4.2.1 Rate Limiting API Scope & Limitations

Currently, the Rate Limiting APIs are only callable from kernel mode entities. Rate Limiting APIs are only supported from package version 2.3.0.x onwards

CHAPTER

FIVE

RATE LIMITING API REFERENCE

5.1 Rate Limiting Functions

5.1.1 cpaGetDevRlPropertiesHandle

This function is used to get the rate limiting properties handle associated with a device for a specified service type. An application could use cpaGetNumDevices to discover number of devices on the system and use cpaGetDeviceInfo to discover device information for a given device index, before using an index to retrieve the rate limiting properties handle.

Syntax:

```
CpaStatus
cpaGetDevRlPropertiesHandle(
   Cpa16U devIdx,
   const CpaAccelerationServiceType svcType,
   CpaRlPropertiesHandle* handle,
   CpaRlError *rlError
);
```

Parameters:

[in] Cpa16U devIdx

Device index from which to fetch the rate limiting handle. The valid range of indices can be discovered by cpaGetNumDevices function.

[in] const CpaAccelerationServiceType svcType

The service type that the rate limiting handle is associated with.

[out] CpaRlPropertiesHandle* handle

Pointer to a rate limiting handle for service specified on the device.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the rate limiting handle.

continues on next page

Table 5.1 – continued from previous page

Return Code	Description
CPA_STATUS_FAIL	The function failed to retrieve the rate limiting handle. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.2 cpaRlPropGetNumInterfaces

This function is used to query the number of interfaces that can be assigned an SLA on a device associated with the rate limiting handle, including the total number of interfaces and the number of interfaces currently unassigned. The rate limiting handle used here should be obtained using cpaGetDevRlPropertiesHandle function.

Syntax:

```
CpaStatus
cpaRlPropGetNumInterfaces(
   const CpaRlPropertiesHandle handle,
   Cpa32U *totalSlaInterfaces,
   Cpa32U *remSlaInterfaces,
   CpaRlError *rlError
);
```

Parameters:

[in] const CpaRlPropertiesHandle handle

Rate limiting handle that specifies a service on a device.

[out] Cpa32U *totalSlaInterfaces

Total number of interfaces that can be assigned an SLA on this device for the service that is represented by the handle.

[out] Cpa32U *remSlaInterfaces

Total number of unassigned SLA interfaces on this device for the service that is represented by the handle.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the total number of interfaces and the unassigned interfaces.

continues on next page

Table 5.2 – continued from previous page

Return Code	Description
CPA_STATUS_FAIL	The function failed to retrieve the interface numbers. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.3 cpaGetDevRlPropSlaCir

This function is used to query the CIR values of the service that is represented by the rate limiting handle, including the total CIR and remaining CIR. The rate limiting handle used here should be obtained using cpaGetDevRlPropertiesHandle function.

The returned value of CIRs are absolute value in Mbps or Kops. For dc, the unit is Mbps. For asym, the unit is Kops.

Syntax:

```
CpaStatus
cpaGetDevRlPropSlaCir(
   const CpaRlPropertiesHandle handle,
   Cpa32U *totalCir,
   Cpa32U *remCir,
   CpaRlError *rlError
);
```

Parameters:

[in] const CpaRlPropertiesHandle handle

Rate limiting handle that specifies a service on a device.

[out] Cpa32U *totalCir

Total CIR set on the device with the service represented by the handle.

[out] Cpa32U *remCir

Remaining CIR on a device with the service represented by the handle.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the total CIR and remaining CIR.

continues on next page

Table 5.3 – continued from previous page

Return Code	Description
CPA_STATUS_FAIL	The function failed to retrieve the total CIR and remaining CIR. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.4 cpaGetDevRlPropSlaPir

This function is used to query the PIR values of the service that is represented by the rate limiting handle, including the total PIR and the total PIR that is assigned to QPs on this device. It is possible for the total assigned PIR value to exceed the total PIR that can be achieved on the device, because it ss the sum of all the PIRs assigned to QPs, and all the QPs are not expected to perform at their PIR at the same time. The rate limiting handle used here should be obtained using *cpaGetDevRlPropertiesHandle* function.

The returned value of CIRs are absolute value in Mbps or Kops. For dc, the unit is Mbps. For asym, the unit is Kops.

Syntax:

```
CpaStatus
cpaGetDevRlPropSlaPir(
   const CpaRlPropertiesHandle handle,
   Cpa32U *totalPir,
   Cpa32U *totalAssignedPir,
   CpaRlError *rlError
);
```

Parameters:

[in] const CpaRlPropertiesHandle handle

Rate limiting handle that specifies a service on a device.

[out] Cpa32U *totalPir

Total PIR set on the device with the service represented by the handle.

[out] Cpa32U *totalAssignedPir

Total assigned PIR on a device with the service represented by the handle.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the PIR and the total assigned PIR.

continues on next page

Table 5.4 – continued from previous page

Return Code	Description
CPA_STATUS_FAIL	The function failed to retrieve the interface numbers. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.5 cpaGetInstanceRlSlaPir

This function is used to query the PIR of an instance in the guest.

The returned value of CIRs are absolute value in Mbps or Kops. For dc, the unit is Mbps. For asym, the unit is Kops.

Syntax:

```
CpaStatus
cpaGetInstanceRlSlaPir(
   const CpaInstanceHandle handle,
   Cpa32U *pirSetting,
   CpaRlError *rlError
);
```

Parameters:

[in] const CpaInstanceHandle handle

A service instance handle.

[out] Cpa32U *pirSetting

Total PIR set on the instance with the service represented by the handle.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the PIR on the instance.
CPA_STATUS_FAIL	The function failed to retrieve the PIR on the instance. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called inside the partition where the instance is assigned.

5.1.6 cpaGetInstanceRlSlaCir

This function is used to query the CIR of an instance.

The returned value of CIRs are absolute value in Mbps or Kops. For dc, the unit is Mbps. For asym, the unit is Kops.

Syntax:

```
CpaStatus
cpaGetInstanceRlSlaCir(
   const CpaInstanceHandle handle,
   Cpa32U *cirSetting,
   CpaRlError *rlError
);
```

Parameters:

[in] const CpaInstanceHandle handle

A service instance handle.

[out] Cpa32U *cirSetting

The CIR set on the instance with the service represented by the handle.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the CIR on the instance.
CPA_STATUS_FAIL	The function failed to retrieve the CIR on the instance. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called inside the partition where the instance is assigned.

5.1.7 cpaRlGetQpNumHandles

This function is used in the host to get the number of QP handles associated with a device for a given service type.

Syntax:

```
CpaStatus
cpaRlGetQpNumHandles(
   Cpa32U devIdx,
   CpaAccelerationServiceType svcType,
   Cpa8U* numHandles,
   CpaRlError *rlError
);
```

Parameters:

[in] Cpa32U devIdx

The device index to fetch the number of QP handles from.

[out] CpaAccelerationServiceType svcType

The service type of the QP handles.

[out] Cpa8U* numHandles

The number of QP handles associated with the service type on the specific device.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the total number of QP handles with the specified service type on the device.
CPA_STATUS_FAIL	The function failed to retrieved the number of QP handles with the specified service type on the device. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.8 cpaRlGetQpHandles

This function is used in the host to get the QP handles associated with a device for a given service type. These QP handles are for the host application to use for managing SLA settings. The user needs to allocate an array as an input to be populated by this function. Before calling this function, the user needs to call cpaRlGetQpNumHandles to get the size of the array for holding the handles.

Syntax:

```
CpaStatus
cpaRlGetQpHandles(
   Cpa32U devIdx,
   CpaAccelerationServiceType svcType,
   CpaRlQpHandle* handles,
   CpaRlError *rlError
);
```

Parameters:

[in] Cpa32U devIdx

The device index to fetch the QP handles from.

[out] CpaAccelerationServiceType svcType

The service type of the QP handles.

[out] CpaRlQpHandle* handles

Pointer to the user-allocated array where the QP handles will be written.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the QP handles associated with the specified service type on the device.
CPA_STATUS_FAIL	The function failed to retrieve the QP handles associated with the specified service type on the device. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.9 cpaSetRlSla

This function is used in the host to set the SLA information on the given QP handle. The QP handle is retrieved by calling cpaRlGetQpHandles. The user needs to populate the CpaUserSla structure with PIR and CIR settings as an input for this function. The PIR value cannot exceed the PIR value for the specific service on the device. The CIR value cannot exceed the remaining CIR value for the specific service on the device.

Syntax:

```
CpaStatus
cpaSetRlSla(
   CpaRlQpHandle handle,
   CpaUserSla *sla,
   CpaRlError *rlError
);
```

Parameters:

[in] CpaRlQpHandle handle

Handle to the QP to which the SLA will be set.

[in] CpaUserSla *sla

SLA information to be set.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully set the SLA on the QP represented by the handle.
CPA_STATUS_FAIL	The function failed to set the SLA on the QP represented by the handle. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device. - CPA_STATUS_RL_INVALID_CIR_PIR, if CIR or PIR requested exceeds device available capacity. - CPA_STATUS_RL_FAIL_IO, failed to send request to hardware.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.
CPA_STATUS_RESOURCE	Error related to system resources.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.10 cpaDeleteRlSla

This function is used in the host to delete the SLA information on the given QP handle. The QP handle is retrieved by calling cpaRlGetQpHandles. The QP should have an existing SLA configuration to be deleted. Once the SLA is deleted on this QP, the bandwidth for this QP is reduced to 0.

Syntax:

```
CpaStatus
cpaDeleteRlSla(
   CpaRlQpHandle handle,
   CpaRlError *rlStatus
);
```

Parameters:

[in] CpaRlQpHandle handle

Handle to the QP to which the SLA will be deleted.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully deleted the SLA on the QP represented by the handle.
CPA_STATUS_FAIL	The function failed to delete the SLA on the QP represented by the handle. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device. - CPA_STATUS_RL_SLA_NOT_CONFIGURED, SLA is not configured on this QP. - CPA_STATUS_RL_FAIL_IO, failed to send request to hardware.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.
CPA_STATUS_RESOURCE	Error related to system resources.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.11 cpaGetRlSla

This function is used in the host to get the SLA information on the given QP handle. The QP handle is retrieved by calling cpaRlGetQpHandles. The QP should have an existing SLA configuration. User needs to allocate the CpaUserSla structure to be populated.

Syntax:

```
CpaStatus
cpaGetRlSla(
   const CpaRlQpHandle handle,
   CpaUserSla *sla,
   CpaRlError *rlError
);
```

Parameters:

[in] CpaRlQpHandle handle

Handle to the QP to which the SLA will be retrieved.

[out] CpaUserSla *sla

SLA information gotten from QP.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully retrieved the SLA configuration on the QP represented by the handle.
CPA_STATUS_FAIL	The function failed to retrieve the SLA configuration on the QP represented by the handle. Check rlError for RL specific status code: - CPA_STATUS_RL_NOT_ENABLED, rate limiting is not enabled on this device. - CPA_STATUS_RL_SLA_NOT_CONFIGURED, SLA is not configured on this QP. - CPA_STATUS_RL_FAIL_IO, failed to send request to hardware.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.
CPA_STATUS_RESOURCE	Error related to system resources.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.12 cpaEnableRateLimiting

This function is used in the host to enable rate limiting on a device if it is not already enabled. If the rate limiting is already enabled, then the function would return successfully.

Syntax:

```
CpaStatus
cpaEnableRateLimiting(
   Cpa32U devIdx,
   CpaRlError *rlError
);
```

Parameters:

[in] Cpa32U devIdx

Index of the device to enable rate limiting on.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully enabled rate limiting on the specified device.
CPA_STATUS_FAIL	The function failed to enable rate limiting on the given device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.1.13 cpaDisableRateLimiting

This function is used in the host to disable rate limiting on a device if it is already enabled. If the rate limiting is already disabled, then the function would return successfully.

Syntax:

```
CpaStatu
cpaDisableRateLimiting(
   Cpa32U devIdx,
   CpaRlError *rlError
);
```

Parameters:

[in] Cpa32U devIdx

Index of the device to disable rate limiting on.

[out] CpaRlError *rlError

Error status of the rate limiting query. This parameter is only valid if CPA_STATUS_FAIL is returned.

Return Value:

Return Code	Description
CPA_STATUS_SUCCESS	The function successfully disabled rate limiting on the specified device.
CPA_STATUS_FAIL	The function failed to disable rate limiting on the given device.
CPA_STATUS_INVALID_PARAM	One or more parameters are invalid.
CPA_STATUS_UNSUPPORTED	Rate limiting feature is not supported on this platform.

Remarks:

This function can only be called from the partition that has the physical function (PF) device mapped.

5.2 Rate Limiting Data Structures

5.2.1 CpaRlError

This enum is used as a parameter in the rate limiting APIs to support additional details about the rate limiting requests status that is not captured by the generic cpa error code when function returns *CPA_STATUS_FAIL*.

Syntax:

```
typedef enum _CpaRlError
{
   CPA_STATUS_RL_FAIL = -1,
   CPA_STATUS_RL_FAIL_IO = -2,
   CPA_STATUS_RL_NOT_ENABLED = -3,
   CPA_STATUS_RL_INVALID_CIR_PIR = -4,
   CPA_STATUS_RL_SLA_NOT_CONFIGURED = -5,
} CpaRlError;
```

Value	Description
CPA_STATUS_RL_FAIL	Failure not covered in the other cases.
CPA_STATUS_RL_FAIL_IO	Failed to send request to hardware.
CPA_STATUS_RL_NOT_ENABLED	Rate limiting is not enabled.
CPA_STATUS_RL_INVALID_CIR_PIR	CIR or PIR is beyond the parent node capacity.
CPA_STATUS_RL_SLA_NOT_CONFIG	For SLA delete, SLA is not configured on the QP.

5.2.2 CpaRlQpHandle

Handle used to uniquely identify a QP to configure sla. It is opaque to user.

Definition:

```
typedef void* CpaRlQpHandle;
```

5.2.3 CpaRlPropertiesHandle

Handle used to identify a specific device's rate limiting properties for a specific service. The properties includes available and remaining number of interfaces for SLA configuration, PIR and CIR. It is opaque to user.

Definition:

```
typedef void* CpaRlPropertiesHandle;
```

5.2.4 CpaUserSla

The user SLA info structure. This structure on the user side to set or save SLA information. For setting an SLA on a QP, the user needs to allocate and populate an SLA structure as an input parameter for the setter function. For querying SLA info on a QP, the user needs to allocate an SLA structure as an output parameter for the getter function.

The structure contains the CIR, PIR and service type for a QP handle. The value of CIR and PIR are absolute value in Mbps or Kops. The unit is dependent on the service type as following:

- Dc: Mbps
- Asym: Kops

Syntax:

```
typedef struct _CpaUserSla {
   CpaAccelerationServiceType svcType;
   Cpa32U cir;
   Cpa32U pir;
} CpaUserSla;
```

Members:

Member	Description
CpaAccelerationServiceType svcType	Service type associated with the cpaQpHandle. Supported service types are: • CPA_ACC_SVC_TYPE_DATA_COMPRESSION • CPA_ACC_SVC_TYPE_CRYPTO_ASYM • CPA_ACC_SVC_TYPE_CRYPTO_SYM This parameter is only for checking, not a setter param.
Cpa32U cir	Committed Information Rate for the associated cpaQpHandle. With a correctly provisioned configuration, a QP should always be able to use acceleration bandwidth up to its CIR.
Cpa32U pir	Peak Information Rate for the associated cpaQpHandle. The handle's rate must not exceed this. PIR is always greater than or equal to CIR.