Intel® QuickAssist Technology Compression API Reference

Automatically generated from sources, June 11, 2014.

Based on API version 1.3

(See Release Notes to map API version to software package version.)

Reference Number: 330686-001

Reference Number: 330686-001

Revision History

Date	Revision	Description
June 2014	001	First public version of the document. Based on Intel Confidential document number 410926-1.3 with the revision history of that document retained for reference purposes.
February 2013	1.3	· Supports supplying multiple intermediate buffer lists when starting a compression instance. Also provides a utility function to determine the number of intermediate buffer lists required by an implementation. · API extensions to support endOfLastBlock detection within a deflate stream.
January 2013	1.2	Resolves the following work requests: · TECG00000185: Changing use of flush flags for stateless compression. Adding support for passing an initial checksum into a stateless compression request. Adding a constraint that cpaDcGenerateFooter() is not supported for stateless operations.
November 2012	1.1	Resolves the following work requests: · TECG00000189: Add a unique instance identifier to CpaInstanceInfo2 · TECG00000193: Enhanced auto select best
October 2012	1.0	Resolves the following work requests: · TECG00000186: Add instance notification support for RESTARTING & RESTARTED events and CPA_STATUS_RESTARTING return codes.
June 2012	0.93	Resolved review comments against previous version which resulted in minor updates to the API comments. Resolved the following work requests: · TECG00000179: Adding version number to compression API
May 2012	0.92	Resolved the following work requests: · TECG00000172: Remove references to cpaDcSessionCreate in cpa_dc.h · TECG00000170: cpaCySymDpSessionCtxGetSize() returns a fixed value · TECG00000173 and TECG00000174 updates/cleanup of api comments · TECG00000174: Updated checksum processing rules.
March 2012	0.92RC6	Added -12 and -13 error codes
March 2012	0.92RC7	Resolved the following work requests: · TECG00000166: Added ability to query bus address information for a CpaInstance.
November 2011	0.92RC5	Added internal memory store to capabilities reporting
September 2011	0.92RC4	Addressed review comments
July 2011	0.92RC3	Completed data plane API

		· Moved results structure to 1st 64 byte section · Added buffer sizes for use by driver
May 2011	0.92RC2	Addressed comments in data plane API
March 2011	0.92RC1	Added data plane API
October 2010	0.91RC2	Minor typo fixes
September 2010	0.91RC1	Based on feedback, incorporated the following: · Converted statistics counters to 64 bit · Improved the results structure · Updated memory configuration for consistency with other services
March 2010	0.9RC5	Based on review and feedback, incorporated the following: • Added a results structure to the compress and decompress interfaces, and to the callback API • added enums to define the potential failures of the accelerators • Intermediate buffer is now a buffer list.
January 2010	0.9RC4-2	Added size of context field to cpaDcGetSessionSize
December 2009	0.90RC4	Based on feedback, incorporated the following: • Separated checksum algorithms in capabilities • Added return code CPA_DC_BAD_DATA return code • Bundled return information to include bytes consumed, bytes produced and checksum • Clean up of some documentation
Sept 21 2009	0.90RC3	Updated as a result of review, incorporate the following changes; • Compression window capabilities now split for compress and decompress. • Update statistic to be more consistent with other APIs. • Added pHistoryBuffer to support state-full deflate. • Removed reference to having different instances able to process the same session.
July 2009	0.90RC2	Added distinction in capabilities for stateful and stateless, compression and decompression Replaced cpaPmGetInstanceInfo with cpaPmGetInstanceInfo2 that gets a new info structure, CpaInstanceInfo2, which supersedes the previous version. Additional info includes physical id, core affinity, and NUMA relevant node.
June 2009	0.90RC1	Added capabilities Add distinction between stateful and stateless. Updated with cpaDcGet/SetMemoryConfiguration Changed from buffer lists to u32 pointers for responses.
February 2009	0.74	1. Addition of response Arguments - APIs can use source and destination buffers in an easier fashion 2. Change from flat buffers to buffer lists to align with QA conventions 3. Major re-write of description of buffer rules 4. Addition of dynamic Huffman trees 5. Removal of file based functions. It was deemed that this functionality could be built using other buffer based APIs

		6. Clean up of session parameters and various typos
December 2008	0.73	First released version of this document with new generation process.

Table of Contents

Table of Contents

1 Deprecated List

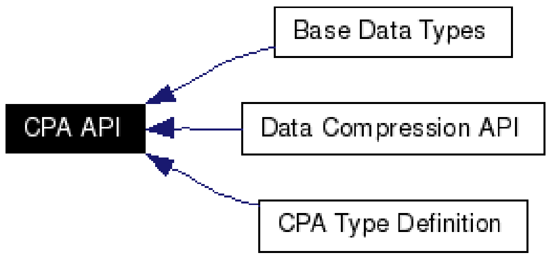
Class **_CpaInstanceInfo**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceInfo2.

Global **CPA_DEPRECATED**
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaAccelerationServiceType**.

Global **CPA_DEPRECATED**
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaOperationalState**.

2 CPA API

Collaboration diagram for CPA API:



2.1 Detailed Description

File: cpa.h

This is the top level API definition for Intel(R) QuickAssist Technology. It contains structures, data types and definitions that are common across the interface.

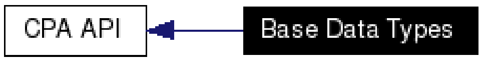
2.2 Modules

- **Base Data Types**
- **CPA Type Definition**
- **Data Compression API**

3 Base Data Types

[CPA API]

Collaboration diagram for Base Data Types:



3.1 Detailed Description

File: cpa.h

The base data types for the Intel CPA API.

3.2 Data Structures

- struct **_CpaFlatBuffer**
- struct **_CpaBufferList**
- struct **_CpaPhysFlatBuffer**
- struct **_CpaPhysBufferList**
- struct **_CpaInstanceInfo**
- struct **_CpaPhysicalInstanceId**
- struct **_CpaInstanceInfo2**

3.3 Defines

- #define **CPA_INSTANCE_HANDLE_SINGLE**
- #define **CPA_DP_BUFLIST**
- #define **CPA_STATUS_SUCCESS**
  Success status value.
- #define **CPA_STATUS_FAIL**
  Fail status value.
- #define **CPA_STATUS_RETRY**
  Retry status value.
- #define **CPA_STATUS_RESOURCE**
  The resource that has been requested is unavailable.
- #define **CPA_STATUS_INVALID_PARAM**
  Invalid parameter has been passed in.
- #define **CPA_STATUS_FATAL**
  A serious error has occurred.
- #define **CPA_STATUS_UNSUPPORTED**
  The function is not supported, at least not with the specific parameters supplied.
- #define **CPA_STATUS_RESTARTING**
  The API implementation is restarting.
- #define **CPA_STATUS_MAX_STR_LENGTH_IN_BYTES**
  Maximum length of the Overall Status String (including generic and specific strings returned by calls to cpaXxGetStatusText).
- #define **CPA_STATUS_STR_SUCCESS**
  Status string for **CPA_STATUS_SUCCESS**.
- #define **CPA_STATUS_STR_FAIL**

Status string for **CPA_STATUS_FAIL**.
- #define **CPA_STATUS_STR_RETRY**
  Status string for **CPA_STATUS_RETRY**.
- #define **CPA_STATUS_STR_RESOURCE**
  Status string for **CPA_STATUS_RESOURCE**.
- #define **CPA_STATUS_STR_INVALID_PARAM**
  Status string for **CPA_STATUS_INVALID_PARAM**.
- #define **CPA_STATUS_STR_FATAL**
  Status string for **CPA_STATUS_FATAL**.
- #define **CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES**
  Maximum instance info name string length in bytes.
- #define **CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES**
  Maximum instance info id string length in bytes.
- #define **CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES**
  Maximum instance info version string length in bytes.

3.4 Typedefs

- typedef void * **CpaInstanceHandle**
- typedef **Cpa64U** **CpaPhysicalAddr**
- typedef **CpaPhysicalAddr**(* **CpaVirtualToPhysical**)(void *pVirtualAddr)
- typedef **_CpaFlatBuffer** **CpaFlatBuffer**
- typedef **_CpaBufferList** **CpaBufferList**
- typedef **_CpaPhysFlatBuffer** **CpaPhysFlatBuffer**
- typedef **_CpaPhysBufferList** **CpaPhysBufferList**
- typedef **Cpa32S** **CpaStatus**
- typedef enum **_CpaInstanceType** **CPA_DEPRECATED**
- typedef enum **_CpaAccelerationServiceType** **CpaAccelerationServiceType**
- typedef enum **_CpaInstanceState** **CPA_DEPRECATED**
- typedef enum **_CpaOperationalState** **CpaOperationalState**
- typedef **_CpaInstanceInfo** **CPA_DEPRECATED**
- typedef **_CpaPhysicalInstanceId** **CpaPhysicalInstanceId**
- typedef **_CpaInstanceInfo2** **CpaInstanceInfo2**
- typedef enum **_CpaInstanceEvent** **CpaInstanceEvent**

3.5 Enumerations

- enum **_CpaInstanceType** {
  CPA_INSTANCE_TYPE_CRYPTO,
  CPA_INSTANCE_TYPE_DATA_COMPRESSION,
  CPA_INSTANCE_TYPE_RAID,
  CPA_INSTANCE_TYPE_XML,
  CPA_INSTANCE_TYPE_REGEX
  }
- enum **_CpaAccelerationServiceType** {
  CPA_ACC_SVC_TYPE_CRYPTO,
  CPA_ACC_SVC_TYPE_DATA_COMPRESSION,
  CPA_ACC_SVC_TYPE_PATTERN_MATCH,
  CPA_ACC_SVC_TYPE_RAID,
  CPA_ACC_SVC_TYPE_XML,
  CPA_ACC_SVC_TYPE_VIDEO_ANALYTICS
  }
- enum **_CpaInstanceState** {
  CPA_INSTANCE_STATE_INITIALISED,
  CPA_INSTANCE_STATE_SHUTDOWN

}
- enum **_CpaOperationalState** {
  CPA_OPER_STATE_DOWN,
  CPA_OPER_STATE_UP
  }
- enum **_CpaInstanceEvent** {
  CPA_INSTANCE_EVENT_RESTARTING,
  CPA_INSTANCE_EVENT_RESTARTED
  }

3.6 Data Structure Documentation

3.6.1 _CpaFlatBuffer Struct Reference

3.6.1.1 Detailed Description

File: cpa.h

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address. An API instance may require the actual data to be in contiguous physical memory as determined by **CpaInstanceInfo2**.

3.6.1.2 Data Fields

- **Cpa32U dataLenInBytes**
  Data length specified in bytes.
- **Cpa8U * pData**
  The data pointer is a virtual address, however the actual data pointed to is required to be in contiguous physical memory unless the field requiresPhysicallyContiguousMemory in CpaInstanceInfo2 is false.

3.6.1.3 Field Documentation

Cpa32U _CpaFlatBuffer::dataLenInBytes

Data length specified in bytes.

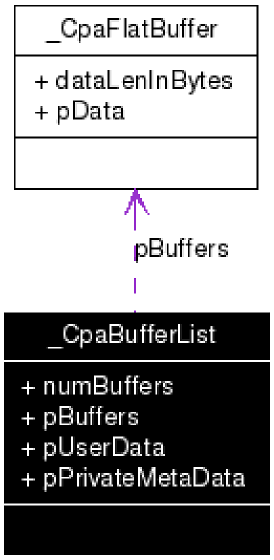
When used as an input parameter to a function, the length specifies the current length of the buffer. When used as an output parameter to a function, the length passed in specifies the maximum length of the buffer on return (i.e. the allocated length). The implementation will not write past this length. On return, the length is always unchanged.

Cpa8U* _CpaFlatBuffer::pData

The data pointer is a virtual address, however the actual data pointed to is required to be in contiguous physical memory unless the field requiresPhysicallyContiguousMemory in CpaInstanceInfo2 is false.

3.6.2 _CpaBufferList Struct Reference

Collaboration diagram for _CpaBufferList:



3.6.2.1 Detailed Description

File: cpa.h

Scatter/Gather buffer list containing an array of flat buffers.

A scatter/gather buffer list structure. This buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous.

Note:

The memory for the pPrivateMetaData member must be allocated by the client as physically contiguous memory. When allocating memory for pPrivateMetaData, a call to the corresponding BufferListGetMetaSize function (e.g. cpaCyBufferListGetMetaSize) MUST be made to determine the size of the Meta Data Buffer. The returned size (in bytes) may then be passed in a memory allocation routine to allocate the pPrivateMetaData memory.

3.6.2.2 Data Fields

- **Cpa32U numBuffers**
  Number of buffers in the list.
- **CpaFlatBuffer** * **pBuffers**
  Pointer to an unbounded array containing the number of CpaFlatBuffers defined by numBuffers.
- void * **pUserData**
  This is an opaque field that is not read or modified internally.
- void * **pPrivateMetaData**
  Private representation of this buffer list.

3.6.2.3 Field Documentation

Cpa32U _CpaBufferList::numBuffers

Number of buffers in the list.

CpaFlatBuffer* _CpaBufferList::pBuffers

Pointer to an unbounded array containing the number of CpaFlatBuffers defined by numBuffers.

void* **_CpaBufferList::pUserData**

This is an opaque field that is not read or modified internally.

void* **_CpaBufferList::pPrivateMetaData**

Private representation of this buffer list.

The memory for this buffer needs to be allocated by the client as contiguous data. The amount of memory required is returned with a call to the corresponding BufferListGetMetaSize function. If that function returns a size of zero then no memory needs to be allocated, and this parameter can be NULL.

3.6.3 _CpaPhysFlatBuffer Struct Reference

3.6.3.1 Detailed Description

File: cpa.h

Flat buffer structure with physical address.

Functions taking this structure do not need to do any virtual to physical address translation before writing the buffer to hardware.

3.6.3.2 Data Fields

- **Cpa32U dataLenInBytes**
  Data length specified in bytes.
- **Cpa32U reserved**
  Reserved for alignment.
- **CpaPhysicalAddr bufferPhysAddr**
  The physical address at which the data resides.

3.6.3.3 Field Documentation

Cpa32U _CpaPhysFlatBuffer::dataLenInBytes

Data length specified in bytes.

When used as an input parameter to a function, the length specifies the current length of the buffer. When used as an output parameter to a function, the length passed in specifies the maximum length of the buffer on return (i.e. the allocated length). The implementation will not write past this length. On return, the length is always unchanged.

Cpa32U _CpaPhysFlatBuffer::reserved

Reserved for alignment.

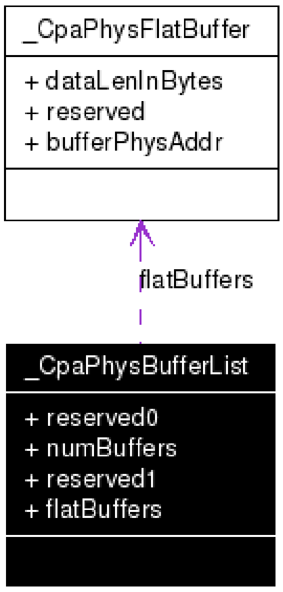
CpaPhysicalAddr _CpaPhysFlatBuffer::bufferPhysAddr

The physical address at which the data resides.

The data pointed to is required to be in contiguous physical memory.

3.6.4 _CpaPhysBufferList Struct Reference

Collaboration diagram for _CpaPhysBufferList:



3.6.4.1 Detailed Description

File: cpa.h

Scatter/gather list containing an array of flat buffers with physical addresses.

Similar to **CpaBufferList**, this buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous. The difference is that, in this case, the individual "flat" buffers are represented using physical, rather than virtual, addresses.

3.6.4.2 Data Fields

- **Cpa64U reserved0**
  Reserved for internal usage.
- **Cpa32U numBuffers**
  Number of buffers in the list.
- **Cpa32U reserved1**
  Reserved for alignment.
- **CpaPhysFlatBuffer flatBuffers** []
  Array of flat buffer structures, of size numBuffers.

3.6.4.3 Field Documentation

Cpa64U _CpaPhysBufferList::reserved0

Reserved for internal usage.

Cpa32U _CpaPhysBufferList::numBuffers

Number of buffers in the list.

Cpa32U _CpaPhysBufferList::reserved1

Reserved for alignment.

CpaPhysFlatBuffer _CpaPhysBufferList::flatBuffers[]

Array of flat buffer structures, of size numBuffers.

3.6.5 _CpaInstanceInfo Struct Reference

3.6.5.1 Detailed Description

File: cpa.h

Instance Info Structure

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceInfo2.

Structure that contains the information to describe the instance.

3.6.5.2 Data Fields

- enum **_CpaInstanceType type**
  Type definition for this instance.
- enum **_CpaInstanceState state**
  Operational state of the instance.
- **Cpa8U name** [CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]
  Simple text string identifier for the instance.
- **Cpa8U version** [CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES]
  Version string.

3.6.5.3 Field Documentation

enum **_CpaInstanceType _CpaInstanceInfo::type**

Type definition for this instance.

enum **_CpaInstanceState _CpaInstanceInfo::state**

Operational state of the instance.

Cpa8U _CpaInstanceInfo::name[CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]

Simple text string identifier for the instance.

Cpa8U _CpaInstanceInfo::version[CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES]

Version string.

There may be multiple versions of the same type of instance accessible through a particular library.

3.6.6 _CpaPhysicalInstanceId Struct Reference

3.6.6.1 Detailed Description

File: cpa.h

Physical Instance ID

Identifies the physical instance of an accelerator execution engine.

Accelerators grouped into "packages". Each accelerator can in turn contain one or more execution engines. Implementations of this API will define the packageId, acceleratorId, executionEngineId and busAddress as appropriate for the implementation. For example, for hardware-based accelerators, the packageId might identify the chip, which might contain multiple accelerators, each of which might contain multiple execution engines. The combination of packageId, acceleratorId and executionEngineId uniquely identifies the instance.

Hardware based accelerators implementing this API may also provide information on the location of the accelerator in the busAddress field. This field will be defined as appropriate for the implementation. For example, for PCIe attached accelerators, the busAddress may contain the PCIe bus, device and function number of the accelerators.

3.6.6.2 Data Fields

- **Cpa16U packageId**
  Identifies the package within which the accelerator is contained.
- **Cpa16U acceleratorId**
  Identifies the specific accelerator within the package.
- **Cpa16U executionEngineId**
  Identifies the specific execution engine within the accelerator.
- **Cpa16U busAddress**
  Identifies the bus address associated with the accelerator execution engine.

3.6.6.3 Field Documentation

Cpa16U _CpaPhysicalInstanceId::packageId

Identifies the package within which the accelerator is contained.

Cpa16U _CpaPhysicalInstanceId::acceleratorId

Identifies the specific accelerator within the package.

Cpa16U _CpaPhysicalInstanceId::executionEngineId

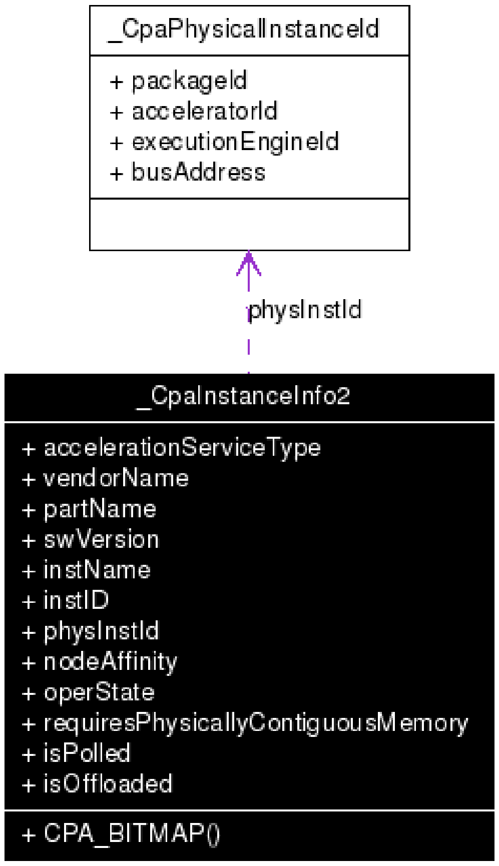
Identifies the specific execution engine within the accelerator.

Cpa16U _CpaPhysicalInstanceId::busAddress

Identifies the bus address associated with the accelerator execution engine.

3.6.7 _CpaInstanceInfo2 Struct Reference

Collaboration diagram for _CpaInstanceInfo2:



3.6.7.1 Detailed Description

File: cpa.h

Instance Info Structure, version 2

Structure that contains the information to describe the instance.

3.6.7.2 Public Member Functions

- **CPA_BITMAP** (coreAffinity, CPA_MAX_CORES)
  A bitmap identifying the core or cores to which the instance is affinitized in an SMP operating system.

3.6.7.3 Data Fields

- **CpaAccelerationServiceType accelerationServiceType**
  Type of service provided by this instance.
- **Cpa8U vendorName** [CPA_INST_VENDOR_NAME_SIZE]
  String identifying the vendor of the accelerator.
- **Cpa8U partName** [CPA_INST_PART_NAME_SIZE]
  String identifying the part (name and/or number).

- **Cpa8U swVersion** [CPA_INST_SW_VERSION_SIZE]
  String identifying the version of the software associated with the instance.
- **Cpa8U instName** [CPA_INST_NAME_SIZE]
  String identifying the name of the instance.
- **Cpa8U instID** [CPA_INST_ID_SIZE]
  String containing a unique identifier for the instance.
- **CpaPhysicalInstanceId physInstId**
  Identifies the "physical instance" of the accelerator.
- **Cpa32U nodeAffinity**
  Identifies the processor complex, or node, to which the accelerator is physically connected, to help identify locality in NUMA systems.
- **CpaOperationalState operState**
  Operational state of the instance.
- **CpaBoolean requiresPhysicallyContiguousMemory**
  Specifies whether the data pointed to by flat buffers (**CpaFlatBuffer::pData**) supplied to this instance must be in physically contiguous memory.
- **CpaBoolean isPolled**
  Specifies whether the instance must be polled, or is event driven.
- **CpaBoolean isOffloaded**
  Identifies whether the instance uses hardware offload, or is a software-only implementation.

3.6.7.4 Member Function Documentation

```
_CpaInstanceInfo2::CPA_BITMAP( coreAffinity    ,
                               CPA_MAX_CORES
                             )
```

A bitmap identifying the core or cores to which the instance is affinitized in an SMP operating system.

The term core here is used to mean a "logical" core - for example, in a dual-processor, quad-core system with hyperthreading (two threads per core), there would be 16 such cores (2 processors x 4 cores/processor x 2 threads/core). The numbering of these cores and the corresponding bit positions is OS-specific. Note that Linux refers to this as "processor affinity" or "CPU affinity", and refers to the bitmap as a "cpumask".

The term "affinity" is used to mean that this is the core on which the callback function will be invoked when using the asynchronous mode of the API. In a hardware-based implementation of the API, this might be the core to which the interrupt is affinitized. In a software-based implementation, this might be the core to which the process running the algorithm is affinitized. Where there is no affinity, the bitmap can be set to all zeroes.

This bitmap should be manipulated using the macros **CPA_BITMAP_BIT_SET**, **CPA_BITMAP_BIT_CLEAR** and **CPA_BITMAP_BIT_TEST**.

3.6.7.5 Field Documentation

CpaAccelerationServiceType _CpaInstanceInfo2::accelerationServiceType

Type of service provided by this instance.

Cpa8U _CpaInstanceInfo2::vendorName[CPA_INST_VENDOR_NAME_SIZE]

String identifying the vendor of the accelerator.

Cpa8U _CpaInstanceInfo2::partName[CPA_INST_PART_NAME_SIZE]

String identifying the part (name and/or number).

Cpa8U _CpaInstanceInfo2::swVersion[CPA_INST_SW_VERSION_SIZE]

String identifying the version of the software associated with the instance.

For hardware-based implementations of the API, this should be the driver version. For software-based implementations of the API, this should be the version of the library.

Note that this should NOT be used to store the version of the API, nor should it be used to report the hardware revision (which can be captured as part of the **partName**, if required).

Cpa8U _CpaInstanceInfo2::instName[CPA_INST_NAME_SIZE]

String identifying the name of the instance.

Cpa8U _CpaInstanceInfo2::instID[CPA_INST_ID_SIZE]

String containing a unique identifier for the instance.

CpaPhysicalInstanceId _CpaInstanceInfo2::physInstId

Identifies the "physical instance" of the accelerator.

Cpa32U _CpaInstanceInfo2::nodeAffinity

Identifies the processor complex, or node, to which the accelerator is physically connected, to help identify locality in NUMA systems.

The values taken by this attribute will typically be in the range 0..n-1, where n is the number of nodes (processor complexes) in the system. For example, in a dual-processor configuration, n=2. The precise values and their interpretation are OS-specific.

CpaOperationalState _CpaInstanceInfo2::operState

Operational state of the instance.

CpaBoolean _CpaInstanceInfo2::requiresPhysicallyContiguousMemory

Specifies whether the data pointed to by flat buffers (**CpaFlatBuffer::pData**) supplied to this instance must be in physically contiguous memory.

CpaBoolean _CpaInstanceInfo2::isPolled

Specifies whether the instance must be polled, or is event driven.

For hardware accelerators, the alternative to polling would be interrupts.

CpaBoolean _CpaInstanceInfo2::isOffloaded

Identifies whether the instance uses hardware offload, or is a software-only implementation.

3.7 Define Documentation

#define CPA_INSTANCE_HANDLE_SINGLE

File: cpa.h

Default instantiation handle value where there is only a single instance

Used as an instance handle value where only one instance exists.

#define CPA_DP_BUFLIST

File: cpa.h

Special value which can be taken by length fields on some of the "data plane" APIs to indicate that the buffer in question is of type CpaPhysBufferList, rather than simply an array of bytes.

#define CPA_STATUS_SUCCESS

Success status value.

#define CPA_STATUS_FAIL

Fail status value.

#define CPA_STATUS_RETRY

Retry status value.

#define CPA_STATUS_RESOURCE

The resource that has been requested is unavailable.

Refer to relevant sections of the API for specifics on what the suggested course of action is.

#define CPA_STATUS_INVALID_PARAM

Invalid parameter has been passed in.

#define CPA_STATUS_FATAL

A serious error has occurred.

Recommended course of action is to shutdown and restart the component.

#define CPA_STATUS_UNSUPPORTED

The function is not supported, at least not with the specific parameters supplied.

This may be because a particular capability is not supported by the current implementation.

#define CPA_STATUS_RESTARTING

The API implementation is restarting.

This may be reported if, for example, a hardware implementation is undergoing a reset. Recommended course of action is to retry the request.

#define CPA_STATUS_MAX_STR_LENGTH_IN_BYTES

Maximum length of the Overall Status String (including generic and specific strings returned by calls to cpaXxGetStatusText).

File: cpa.h

API status string type definition

This type definition is used for the generic status text strings provided by cpaXxGetStatusText API functions. Common values are defined, for example see **CPA_STATUS_STR_SUCCESS**, **CPA_STATUS_FAIL**, etc., as well as the maximum size **CPA_STATUS_MAX_STR_LENGTH_IN_BYTES**.

#define CPA_STATUS_STR_SUCCESS

Status string for **CPA_STATUS_SUCCESS**.

#define CPA_STATUS_STR_FAIL

Status string for **CPA_STATUS_FAIL**.

#define CPA_STATUS_STR_RETRY

Status string for **CPA_STATUS_RETRY**.

#define CPA_STATUS_STR_RESOURCE

Status string for **CPA_STATUS_RESOURCE**.

#define CPA_STATUS_STR_INVALID_PARAM

Status string for **CPA_STATUS_INVALID_PARAM**.

#define CPA_STATUS_STR_FATAL

Status string for **CPA_STATUS_FATAL**.

#define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES

Maximum instance info name string length in bytes.

#define CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES

Maximum instance info id string length in bytes.

#define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES

Maximum instance info version string length in bytes.

3.8 Typedef Documentation

typedef void* **CpaInstanceHandle**

File: cpa.h

Instance handle type.

Handle used to uniquely identify an instance.

Note:

Where only a single instantiation exists this field may be set to **CPA_INSTANCE_HANDLE_SINGLE**.

typedef **Cpa64U CpaPhysicalAddr**

File: cpa.h

Physical memory address.

Type for physical memory addresses.

typedef **CpaPhysicalAddr**(* **CpaVirtualToPhysical**)(void *pVirtualAddr)

File: cpa.h

Virtual to physical address conversion routine.

This function is used to convert virtual addresses to physical addresses.

Context:
The function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pVirtualAddr*	Virtual address to be converted.

Returns:
Returns the corresponding physical address. On error, the value NULL is returned.

Postcondition:
None

See also:
None

typedef struct **_CpaFlatBuffer CpaFlatBuffer**

File: cpa.h

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address. An API instance may require the actual data to be in contiguous physical memory as determined by **CpaInstanceInfo2**.

typedef struct **_CpaBufferList CpaBufferList**

File: cpa.h

Scatter/Gather buffer list containing an array of flat buffers.

A scatter/gather buffer list structure. This buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous.

Note:

The memory for the pPrivateMetaData member must be allocated by the client as physically contiguous memory. When allocating memory for pPrivateMetaData, a call to the corresponding BufferListGetMetaSize function (e.g. cpaCyBufferListGetMetaSize) MUST be made to determine the size of the Meta Data Buffer. The returned size (in bytes) may then be passed in a memory allocation routine to allocate the pPrivateMetaData memory.

typedef struct **_CpaPhysFlatBuffer CpaPhysFlatBuffer**

File: cpa.h

Flat buffer structure with physical address.

Functions taking this structure do not need to do any virtual to physical address translation before writing the buffer to hardware.

typedef struct **_CpaPhysBufferList CpaPhysBufferList**

File: cpa.h

Scatter/gather list containing an array of flat buffers with physical addresses.

Similar to **CpaBufferList**, this buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous. The difference is that, in this case, the individual "flat" buffers are represented using physical, rather than virtual, addresses.

typedef **Cpa32S CpaStatus**

File: cpa.h

API status value type definition

This type definition is used for the return values used in all the API functions. Common values are defined, for example see **CPA_STATUS_SUCCESS**, **CPA_STATUS_FAIL**, etc.

typedef enum **_CpaInstanceType CPA_DEPRECATED**

File: cpa.h

Instance Types

Deprecated:

As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaAccelerationServiceType**.

Enumeration of the different instance types.

typedef enum **_CpaAccelerationServiceType CpaAccelerationServiceType**

File: cpa.h

Service Type

Enumeration of the different service types.

typedef enum **_CpaInstanceState CPA_DEPRECATED**

File: cpa.h

Instance State

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaOperationalState**.

Enumeration of the different instance states that are possible.

typedef enum **_CpaOperationalState CpaOperationalState**

File: cpa.h

Instance operational state

Enumeration of the different operational states that are possible.

typedef struct **_CpaInstanceInfo CPA_DEPRECATED**

File: cpa.h

Instance Info Structure

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceInfo2.

Structure that contains the information to describe the instance.

typedef struct **_CpaPhysicalInstanceId CpaPhysicalInstanceId**

File: cpa.h

Physical Instance ID

Identifies the physical instance of an accelerator execution engine.

Accelerators grouped into "packages". Each accelerator can in turn contain one or more execution engines. Implementations of this API will define the packageId, acceleratorId, executionEngineId and busAddress as appropriate for the implementation. For example, for hardware-based accelerators, the packageId might identify the chip, which might contain multiple accelerators, each of which might contain multiple execution engines. The combination of packageId, acceleratorId and executionEngineId uniquely identifies the instance.

Hardware based accelerators implementing this API may also provide information on the location of the accelerator in the busAddress field. This field will be defined as appropriate for the implementation. For example, for PCIe attached accelerators, the busAddress may contain the PCIe bus, device and function number of the accelerators.

typedef struct **_CpaInstanceInfo2 CpaInstanceInfo2**

File: cpa.h

Instance Info Structure, version 2

Structure that contains the information to describe the instance.

typedef enum **_CpaInstanceEvent CpaInstanceEvent**

File: cpa.h

Instance Events

Enumeration of the different events that will cause the registered Instance notification callback function to be invoked.

3.9 Enumeration Type Documentation

enum **_CpaInstanceType**

File: cpa.h

Instance Types

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaAccelerationServiceType**.

Enumeration of the different instance types.

Enumerator:

CPA_INSTANCE_TYPE_CRYPTO	Cryptographic instance type.
CPA_INSTANCE_TYPE_DATA_COMPRESSION	Data compression instance type.
CPA_INSTANCE_TYPE_RAID	RAID instance type.
CPA_INSTANCE_TYPE_XML	XML instance type.
CPA_INSTANCE_TYPE_REGEX	Regular Expression instance type.

enum **_CpaAccelerationServiceType**

File: cpa.h

Service Type

Enumeration of the different service types.

Enumerator:

CPA_ACC_SVC_TYPE_CRYPTO	Cryptography.
CPA_ACC_SVC_TYPE_DATA_COMPRESSION	Data Compression.
CPA_ACC_SVC_TYPE_PATTERN_MATCH	Pattern Match.
CPA_ACC_SVC_TYPE_RAID	RAID.

CPA_ACC_SVC_TYPE_XML	XML.
CPA_ACC_SVC_TYPE_VIDEO_ANALYTICS	Video Analytics.

enum **_CpaInstanceState**

File: cpa.h

Instance State

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaOperationalState**.

Enumeration of the different instance states that are possible.

Enumerator:

CPA_INSTANCE_STATE_INITIALISED	Instance is in the initialized state and ready for use.
CPA_INSTANCE_STATE_SHUTDOWN	Instance is in the shutdown state and not available for use.

enum **_CpaOperationalState**

File: cpa.h

Instance operational state

Enumeration of the different operational states that are possible.

Enumerator:

CPA_OPER_STATE_DOWN	Instance is not available for use. May not yet be initialized, or stopped.
CPA_OPER_STATE_UP	Instance is available for use. Has been initialized and started.

enum **_CpaInstanceEvent**

File: cpa.h

Instance Events

Enumeration of the different events that will cause the registered Instance notification callback function to be invoked.

Enumerator:

CPA_INSTANCE_EVENT_RESTARTING	Event type that triggers the registered instance notification callback function when and instance is restarting. The reason why an instance is restarting is implementation specific. For example a hardware

	implementation may send this event if the hardware device is about to be reset.
CPA_INSTANCE_EVENT_RESTARTED	Event type that triggers the registered instance notification callback function when and instance has restarted. The reason why an instance has restarted is implementation specific. For example a hardware implementation may send this event after the hardware device has been reset.

4 CPA Type Definition

[CPA API]

Collaboration diagram for CPA Type Definition:

4.1 Detailed Description

File: cpa_types.h

This is the CPA Type Definitions.

4.2 Defines

- #define **NULL**
- #define **TRUE**
- #define **FALSE**
- #define **CPA_BITMAP**(name, sizeInBits)
- #define **CPA_BITMAP_BIT_TEST**(bitmask, bit)
  Test a specified bit in the specified bitmap.
- #define **CPA_BITMAP_BIT_SET**(bitmask, bit)
- #define **CPA_BITMAP_BIT_CLEAR**(bitmask, bit)
  Clear a specified bit in the specified bitmap.
- #define **CPA_DEPRECATED**
  Declare a function or type and mark it as deprecated so that usages get flagged with a warning.

4.3 Typedefs

- typedef uint8_t **Cpa8U**
- typedef int8_t **Cpa8S**
- typedef uint16_t **Cpa16U**
- typedef int16_t **Cpa16S**
- typedef uint32_t **Cpa32U**
- typedef int32_t **Cpa32S**
- typedef uint64_t **Cpa64U**
- typedef int64_t **Cpa64S**
- typedef enum **_CpaBoolean CpaBoolean**

4.4 Enumerations

- enum **_CpaBoolean** {
  CPA_FALSE,
  CPA_TRUE
  }

4.5 Define Documentation

```
#define NULL
```

File: cpa_types.h

NULL definition.

```
#define TRUE
```

File: cpa_types.h

True value definition.

```
#define FALSE
```

File: cpa_types.h

False value definition.

```
#define CPA_BITMAP ( name,
                     sizeInBits  )
```

File: cpa_types.h

Declare a bitmap of specified size (in bits).

This macro is used to declare a bitmap of arbitrary size.

To test whether a bit in the bitmap is set, use **CPA_BITMAP_BIT_TEST**.

While most uses of bitmaps on the API are read-only, macros are also provided to set (see **CPA_BITMAP_BIT_SET**) and clear (see **CPA_BITMAP_BIT_CLEAR**) bits in the bitmap.

```
#define CPA_BITMAP_BIT_TEST ( bitmask,
                              bit      )
```

Test a specified bit in the specified bitmap.

The bitmap may have been declared using **CPA_BITMAP**. Returns a Boolean (true if the bit is set, false otherwise).

```
#define CPA_BITMAP_BIT_SET ( bitmask,
                             bit      )
```

File: cpa_types.h

Set a specified bit in the specified bitmap. The bitmap may have been declared using **CPA_BITMAP**.

```
#define CPA_BITMAP_BIT_CLEAR ( bitmask,
                               bit      )
```

Clear a specified bit in the specified bitmap.

The bitmap may have been declared using **CPA_BITMAP**.

#define **CPA_DEPRECATED**

Declare a function or type and mark it as deprecated so that usages get flagged with a warning.

4.6 Typedef Documentation

typedef uint8_t **Cpa8U**

File: cpa_types.h

Unsigned byte base type.

typedef int8_t **Cpa8S**

File: cpa_types.h

Signed byte base type.

typedef uint16_t **Cpa16U**

File: cpa_types.h

Unsigned double-byte base type.

typedef int16_t **Cpa16S**

File: cpa_types.h

Signed double-byte base type.

typedef uint32_t **Cpa32U**

File: cpa_types.h

Unsigned quad-byte base type.

typedef int32_t **Cpa32S**

File: cpa_types.h

Signed quad-byte base type.

typedef uint64_t **Cpa64U**

File: cpa_types.h

Unsigned double-quad-byte base type.

typedef int64_t **Cpa64S**

File: cpa_types.h

Signed double-quad-byte base type.

typedef enum **_CpaBoolean CpaBoolean**

File: cpa_types.h

Boolean type.

Functions in this API use this type for Boolean variables that take true or false values.

4.7 Enumeration Type Documentation

enum **_CpaBoolean**

File: cpa_types.h

Boolean type.

Functions in this API use this type for Boolean variables that take true or false values.

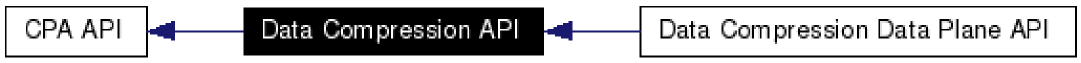
Enumerator:

CPA_FALSE	False value.
CPA_TRUE	True value.

5 Data Compression API

[CPA API]

Collaboration diagram for Data Compression API:



5.1 Detailed Description

File: cpa_dc.h

These functions specify the API for Data Compression operations.

Remarks:

5.2 Modules

- **Data Compression Data Plane API**

5.3 Data Structures

- struct **_CpaDcInstanceCapabilities**
  Implementation Capabilities Structure.
- struct **_CpaDcSessionSetupData**
  Session Setup Data.
- struct **_CpaDcStats**
  Compression Statistics Data.
- struct **_CpaDcRqResults**
  Request results data.

5.4 Defines

- #define **CPA_DC_API_VERSION_NUM_MAJOR**
- #define **CPA_DC_API_VERSION_NUM_MINOR**
- #define **CPA_DC_BAD_DATA**
  Service specific return codesInput data in invalid.

5.5 Typedefs

- typedef void * **CpaDcSessionHandle**
  Compression API session handle type.
- typedef enum **_CpaDcFileType CpaDcFileType**
  Supported file types.
- typedef enum **_CpaDcFlush CpaDcFlush**
  Supported flush flags.
- typedef enum **_CpaDcHuffType CpaDcHuffType**
  Supported Huffman Tree types.
- typedef enum **_CpaDcCompType CpaDcCompType**
  Supported compression types.

- typedef enum **_CpaDcChecksum CpaDcChecksum**
  Supported checksum algorithms.
- typedef enum **_CpaDcSessionDir CpaDcSessionDir**
  Supported session directions.
- typedef enum **_CpaDcSessionState CpaDcSessionState**
  Supported session state settings.
- typedef enum **_CpaDcCompLvl CpaDcCompLvl**
  Supported compression levels.
- typedef enum **_CpaDcReqStatus CpaDcReqStatus**
  Supported additional details from accelerator.
- typedef enum **_CpaDcAutoSelectBest CpaDcAutoSelectBest**
  Supported modes for automatically selecting the best compression type.
- typedef void(* **CpaDcCallbackFn**)(void *callbackTag, **CpaStatus** status)
  Definition of callback function invoked for asynchronous cpaDc requests.
- typedef **_CpaDcInstanceCapabilities CpaDcInstanceCapabilities**
  Implementation Capabilities Structure.
- typedef **_CpaDcSessionSetupData CpaDcSessionSetupData**
  Session Setup Data.
- typedef **_CpaDcStats CpaDcStats**
  Compression Statistics Data.
- typedef **_CpaDcRqResults CpaDcRqResults**
  Request results data.
- typedef void(* **CpaDcInstanceNotificationCbFunc**)(const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, const **CpaInstanceEvent** instanceEvent)
  Callback function for instance notification support.

5.6 Enumerations

- enum **_CpaDcFileType** {
  CPA_DC_FT_ASCII,
  CPA_DC_FT_CSS,
  CPA_DC_FT_HTML,
  CPA_DC_FT_JAVA,
  CPA_DC_FT_OTHER
  }
  Supported file types.
- enum **_CpaDcFlush** {
  CPA_DC_FLUSH_NONE,
  CPA_DC_FLUSH_FINAL,
  CPA_DC_FLUSH_SYNC,
  CPA_DC_FLUSH_FULL
  }
  Supported flush flags.
- enum **_CpaDcHuffType** {
  CPA_DC_HT_STATIC,
  CPA_DC_HT_PRECOMP,
  CPA_DC_HT_FULL_DYNAMIC
  }
  Supported Huffman Tree types.
- enum **_CpaDcCompType** {
  CPA_DC_LZS,
  CPA_DC_ELZS,
  CPA_DC_LZSS,
  CPA_DC_DEFLATE
  }
  Supported compression types.

5.6 Enumerations

- enum **_CpaDcChecksum** {
  CPA_DC_NONE,
  CPA_DC_CRC32,
  CPA_DC_ADLER32
  }

  Supported checksum algorithms.
- enum **_CpaDcSessionDir** {
  CPA_DC_DIR_COMPRESS,
  CPA_DC_DIR_DECOMPRESS,
  CPA_DC_DIR_COMBINED
  }

  Supported session directions.
- enum **_CpaDcSessionState** {
  CPA_DC_STATEFUL,
  CPA_DC_STATELESS
  }

  Supported session state settings.
- enum **_CpaDcCompLvl** {
  CPA_DC_L1,
  CPA_DC_L2,
  CPA_DC_L3,
  CPA_DC_L4,
  CPA_DC_L5,
  CPA_DC_L6,
  CPA_DC_L7,
  CPA_DC_L8,
  CPA_DC_L9
  }

  Supported compression levels.
- enum **_CpaDcReqStatus** {
  CPA_DC_OK,
  CPA_DC_INVALID_BLOCK_TYPE,
  CPA_DC_BAD_STORED_BLOCK_LEN,
  CPA_DC_TOO_MANY_CODES,
  CPA_DC_INCOMPLETE_CODE_LENS,
  CPA_DC_REPEATED_LENS,
  CPA_DC_MORE_REPEAT,
  CPA_DC_BAD_LITLEN_CODES,
  CPA_DC_BAD_DIST_CODES,
  CPA_DC_INVALID_CODE,
  CPA_DC_INVALID_DIST,
  CPA_DC_OVERFLOW,
  CPA_DC_SOFTERR,
  CPA_DC_FATALERR
  }

  Supported additional details from accelerator.
- enum **_CpaDcAutoSelectBest** {
  CPA_DC_ASB_DISABLED,
  CPA_DC_ASB_STATIC_DYNAMIC,
  CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS,
  CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS
  }

  Supported modes for automatically selecting the best compression type.

5.7 Functions

- **CpaStatus cpaDcQueryCapabilities** (**CpaInstanceHandle** dcInstance, **CpaDcInstanceCapabilities** *pInstanceCapabilities)
  Retrieve Instance Capabilities.
- **CpaStatus cpaDcInitSession** (**CpaInstanceHandle** dcInstance, **CpaDcSessionHandle** pSessionHandle, **CpaDcSessionSetupData** *pSessionData, **CpaBufferList** *pContextBuffer, **CpaDcCallbackFn** callbackFn)
  Initialize compression decompression session.
- **CpaStatus cpaDcRemoveSession** (const **CpaInstanceHandle** dcInstance, **CpaDcSessionHandle** pSessionHandle)
  Compression Session Remove Function.
- **CpaStatus cpaDcCompressData** (**CpaInstanceHandle** dcInstance, **CpaDcSessionHandle** pSessionHandle, **CpaBufferList** *pSrcBuff, **CpaBufferList** *pDestBuff, **CpaDcRqResults** *pResults, **CpaDcFlush** flushFlag, void *callbackTag)
  Submit a request to compress a buffer of data.
- **CpaStatus cpaDcDecompressData** (**CpaInstanceHandle** dcInstance, **CpaDcSessionHandle** pSessionHandle, **CpaBufferList** *pSrcBuff, **CpaBufferList** *pDestBuff, **CpaDcRqResults** *pResults, **CpaDcFlush** flushFlag, void *callbackTag)
  Submit a request to decompress a buffer of data.
- **CpaStatus cpaDcGenerateHeader** (**CpaDcSessionHandle** pSessionHandle, **CpaFlatBuffer** *pDestBuff, **Cpa32U** *count)
  Generate compression header.
- **CpaStatus cpaDcGenerateFooter** (**CpaDcSessionHandle** pSessionHandle, **CpaFlatBuffer** *pDestBuff, **CpaDcRqResults** *pResults)
  Generate compression footer.
- **CpaStatus cpaDcGetStats** (**CpaInstanceHandle** dcInstance, **CpaDcStats** *pStatistics)
  Retrieve statistics.
- **CpaStatus cpaDcGetNumInstances** (**Cpa16U** *pNumInstances)
  Get the number of device instances that are supported by the API implementation.
- **CpaStatus cpaDcGetInstances** (**Cpa16U** numInstances, **CpaInstanceHandle** *dcInstances)
  Get the handles to the device instances that are supported by the API implementation.
- **CpaStatus cpaDcGetNumIntermediateBuffers** (**CpaInstanceHandle** instanceHandle, **Cpa16U** *pNumBuffers)
  Compression Component utility function to determine the number of intermediate buffers required by an implementation.
- **CpaStatus cpaDcStartInstance** (**CpaInstanceHandle** instanceHandle, **Cpa16U** numBuffers, **CpaBufferList** **pIntermediateBuffers)
  Compression Component Initialization and Start function.
- **CpaStatus cpaDcInstanceGetInfo2** (const **CpaInstanceHandle** instanceHandle, **CpaInstanceInfo2** *pInstanceInfo2)
  Function to get information on a particular instance.
- **CpaStatus cpaDcInstanceSetNotificationCb** (const **CpaInstanceHandle** instanceHandle, const **CpaDcInstanceNotificationCbFunc** pInstanceNotificationCb, void *pCallbackTag)
  Subscribe for instance notifications.
- **CpaStatus cpaDcGetSessionSize** (**CpaInstanceHandle** dcInstance, **CpaDcSessionSetupData** *pSessionData, **Cpa32U** *pSessionSize, **Cpa32U** *pContextSize)
  Get the size of the memory required to hold the session information.
- **CpaStatus cpaDcBufferListGetMetaSize** (const **CpaInstanceHandle** instanceHandle, **Cpa32U** numBuffers, **Cpa32U** *pSizeInBytes)
  Function to return the size of the memory which must be allocated for the pPrivateMetaData member of CpaBufferList.
- **CpaStatus cpaDcGetStatusText** (const **CpaInstanceHandle** dcInstance, const **CpaStatus** errStatus, **Cpa8S** *pStatusText)
  Function to return a string indicating the specific error that occurred within the system.

- **CpaStatus cpaDcSetAddressTranslation** (const **CpaInstanceHandle** instanceHandle, **CpaVirtualToPhysical** virtual2Physical)
  Set Address Translation function.
- **CpaStatus cpaDcDpGetSessionSize** (**CpaInstanceHandle** dcInstance, **CpaDcSessionSetupData** *pSessionData, **Cpa32U** *pSessionSize)
  Get the size of the memory required to hold the data plane session information.
- **CpaStatus cpaDcDpRemoveSession** (const **CpaInstanceHandle** dcInstance, **CpaDcSessionHandle** pSessionHandle)
  Compression Data Plane Session Remove Function.

5.8 Data Structure Documentation

5.8.1 _CpaDcInstanceCapabilities Struct Reference

5.8.1.1 Detailed Description

Implementation Capabilities Structure.

This structure contains data relating to the capabilities of an implementation. The capabilities include supported compression algorithms, RFC 1951 options and whether the implementation supports both stateful and stateless compress and decompress sessions.

5.8.1.2 Data Fields

- **CpaBoolean statefulLZSCompression**
  True if the Instance supports Stateful LZS compression.
- **CpaBoolean statefulLZSDecompression**
  True if the Instance supports Stateful LZS decompression.
- **CpaBoolean statelessLZSCompression**
  True if the Instance supports Stateless LZS compression.
- **CpaBoolean statelessLZSDecompression**
  True if the Instance supports Stateless LZS decompression.
- **CpaBoolean statefulLZSSCompression**
  True if the Instance supports Stateful LZSS compression.
- **CpaBoolean statefulLZSSDecompression**
  True if the Instance supports Stateful LZSS decompression.
- **CpaBoolean statelessLZSSCompression**
  True if the Instance supports Stateless LZSS compression.
- **CpaBoolean statelessLZSSDecompression**
  True if the Instance supports Stateless LZSS decompression.
- **CpaBoolean statefulELZSCompression**
  True if the Instance supports Stateful Extended LZS compression.
- **CpaBoolean statefulELZSDecompression**
  True if the Instance supports Stateful Extended LZS decompression.
- **CpaBoolean statelessELZSCompression**
  True if the Instance supports Stateless Extended LZS compression.
- **CpaBoolean statelessELZSDecompression**
  True if the Instance supports Stateless Extended LZS decompression.
- **CpaBoolean statefulDeflateCompression**
  True if the Instance supports Stateful Deflate compression.
- **CpaBoolean statefulDeflateDecompression**
  True if the Instance supports Stateful Deflate decompression.
- **CpaBoolean statelessDeflateCompression**
  True if the Instance supports Stateless Deflate compression.

- **CpaBoolean statelessDeflateDecompression**
  True if the Instance supports Stateless Deflate decompression.
- **CpaBoolean checksumCRC32**
  True if the Instance can calculate a CRC32 checksum over the uncompressed data.
- **CpaBoolean checksumAdler32**
  True if the Instance can calculate an Adler-32 checksum over the uncompressed data.
- **CpaBoolean dynamicHuffman**
  True if the Instance supports dynamic Huffman trees in deflate blocks.
- **CpaBoolean dynamicHuffmanBufferReq**
  True if an Instance specific buffer is required to perform a dynamic Huffman tree deflate request.
- **CpaBoolean precompiledHuffman**
  True if the Instance supports precompiled Huffman trees in deflate blocks.
- **CpaBoolean autoSelectBestHuffmanTree**
  True if the Instance has the ability to automatically select between different Huffman encoding schemes for better compression ratios.
- **Cpa8U validWindowSizeMaskCompression**
  Bits set to '1' for each valid window size supported by the compression implementation.
- **Cpa8U validWindowSizeMaskDecompression**
  Bits set to '1' for each valid window size supported by the decompression implementation.
- **Cpa32U internalHuffmanMem**
  Number of bytes internally available to be used when constructing dynamic Huffman trees.
- **CpaBoolean endOfLastBlock**
  True if the Instance supports stopping at the end of the last block in a deflate stream during a decompression operation and reporting that the end of the last block has been reached as part of the CpaDcReqStatus data.

5.8.1.3 Field Documentation

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSCompression

True if the Instance supports Stateful LZS compression.

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSDecompression

True if the Instance supports Stateful LZS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statelessLZSCompression

True if the Instance supports Stateless LZS compression.

CpaBoolean _CpaDcInstanceCapabilities::statelessLZSDecompression

True if the Instance supports Stateless LZS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSSCompression

True if the Instance supports Stateful LZSS compression.

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSSDecompression

True if the Instance supports Stateful LZSS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statelessLZSSCompression

True if the Instance supports Stateless LZSS compression.

CpaBoolean _CpaDcInstanceCapabilities::statelessLZSSDecompression

True if the Instance supports Stateless LZSS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statefulELZSCompression

True if the Instance supports Stateful Extended LZS compression.

CpaBoolean _CpaDcInstanceCapabilities::statefulELZSDecompression

True if the Instance supports Stateful Extended LZS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statelessELZSCompression

True if the Instance supports Stateless Extended LZS compression.

CpaBoolean _CpaDcInstanceCapabilities::statelessELZSDecompression

True if the Instance supports Stateless Extended LZS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statefulDeflateCompression

True if the Instance supports Stateful Deflate compression.

CpaBoolean _CpaDcInstanceCapabilities::statefulDeflateDecompression

True if the Instance supports Stateful Deflate decompression.

CpaBoolean _CpaDcInstanceCapabilities::statelessDeflateCompression

True if the Instance supports Stateless Deflate compression.

CpaBoolean _CpaDcInstanceCapabilities::statelessDeflateDecompression

True if the Instance supports Stateless Deflate decompression.

CpaBoolean _CpaDcInstanceCapabilities::checksumCRC32

True if the Instance can calculate a CRC32 checksum over the uncompressed data.

CpaBoolean _CpaDcInstanceCapabilities::checksumAdler32

True if the Instance can calculate an Adler-32 checksum over the uncompressed data.

CpaBoolean _CpaDcInstanceCapabilities::dynamicHuffman

True if the Instance supports dynamic Huffman trees in deflate blocks.

CpaBoolean _CpaDcInstanceCapabilities::dynamicHuffmanBufferReq

True if an Instance specific buffer is required to perform a dynamic Huffman tree deflate request.

CpaBoolean _CpaDcInstanceCapabilities::precompiledHuffman

True if the Instance supports precompiled Huffman trees in deflate blocks.

CpaBoolean _CpaDcInstanceCapabilities::autoSelectBestHuffmanTree

True if the Instance has the ability to automatically select between different Huffman encoding schemes for better compression ratios.

Cpa8U _CpaDcInstanceCapabilities::validWindowSizeMaskCompression

Bits set to '1' for each valid window size supported by the compression implementation.

Cpa8U _CpaDcInstanceCapabilities::validWindowSizeMaskDecompression

Bits set to '1' for each valid window size supported by the decompression implementation.

Cpa32U _CpaDcInstanceCapabilities::internalHuffmanMem

Number of bytes internally available to be used when constructing dynamic Huffman trees.

CpaBoolean _CpaDcInstanceCapabilities::endOfLastBlock

True if the Instance supports stopping at the end of the last block in a deflate stream during a decompression operation and reporting that the end of the last block has been reached as part of the CpaDcReqStatus data.

5.8.2 _CpaDcSessionSetupData Struct Reference

5.8.2.1 Detailed Description

Session Setup Data.

This structure contains data relating to setting up a session. The client needs to complete the information in this structure in order to setup a session.

5.8.2.2 Data Fields

- **CpaDcCompLvl compLevel**
  Compression Level from CpaDcCompLvl.
- **CpaDcCompType compType**
  Compression type from CpaDcCompType.
- **CpaDcHuffType huffType**
  Huffman type from CpaDcHuffType.
- **CpaDcAutoSelectBest autoSelectBestHuffmanTree**
  Indicates if and how the implementation should select the best Huffman encoding.
- **CpaDcFileType fileType**
  File type for the purpose of determining Huffman Codes from CpaDcFileType.
- **CpaDcSessionDir sessDirection**
  Session direction indicating whether session is used for compression, decompression or both.
- **CpaDcSessionState sessState**
  Session state indicating whether the session should be configured as stateless or stateful.
- **Cpa32U deflateWindowSize**
  Base 2 logarithm of maximum window size minus 8 (a value of 7 for a 32K window size).
- **CpaDcChecksum checksum**
  Desired checksum required for the session.

5.8.2.3 Field Documentation

CpaDcCompLvl _CpaDcSessionSetupData::compLevel

Compression Level from CpaDcCompLvl.

CpaDcCompType _CpaDcSessionSetupData::compType

Compression type from CpaDcCompType.

CpaDcHuffType _CpaDcSessionSetupData::huffType

Huffman type from CpaDcHuffType.

CpaDcAutoSelectBest _CpaDcSessionSetupData::autoSelectBestHuffmanTree

Indicates if and how the implementation should select the best Huffman encoding.

CpaDcFileType _CpaDcSessionSetupData::fileType

File type for the purpose of determining Huffman Codes from CpaDcFileType.

CpaDcSessionDir _CpaDcSessionSetupData::sessDirection

Session direction indicating whether session is used for compression, decompression or both.

CpaDcSessionState _CpaDcSessionSetupData::sessState

Session state indicating whether the session should be configured as stateless or stateful.

Cpa32U _CpaDcSessionSetupData::deflateWindowSize

Base 2 logarithm of maximum window size minus 8 (a value of 7 for a 32K window size).

Permitted values are 0 to 7. cpaDcDecompressData may return an error if an attempt is made to decompress a stream that has a larger window size.

CpaDcChecksum _CpaDcSessionSetupData::checksum

Desired checksum required for the session.

5.8.3 _CpaDcStats Struct Reference

5.8.3.1 Detailed Description

Compression Statistics Data.

This structure contains data elements corresponding to statistics. Statistics are collected on a per instance basis and include: jobs submitted and completed for both compression and decompression.

5.8.3.2 Data Fields

- **Cpa64U numCompRequests**
  Number of successful compression requests.
- **Cpa64U numCompRequestsErrors**
  Number of compression requests that had errors and could not be processed.
- **Cpa64U numCompCompleted**
  Compression requests completed.
- **Cpa64U numCompCompletedErrors**
  Compression requests not completed due to errors.
- **Cpa64U numDecompRequests**
  Number of successful decompression requests.
- **Cpa64U numDecompRequestsErrors**
  Number of decompression requests that had errors and could not be processed.
- **Cpa64U numDecompCompleted**
  Decompression requests completed.
- **Cpa64U numDecompCompletedErrors**
  Decompression requests not completed due to errors.

5.8.3.3 Field Documentation

Cpa64U _CpaDcStats::numCompRequests

Number of successful compression requests.

Cpa64U _CpaDcStats::numCompRequestsErrors

Number of compression requests that had errors and could not be processed.

Cpa64U _CpaDcStats::numCompCompleted

Compression requests completed.

Cpa64U _CpaDcStats::numCompCompletedErrors

Compression requests not completed due to errors.

Cpa64U _CpaDcStats::numDecompRequests

Number of successful decompression requests.

Cpa64U _CpaDcStats::numDecompRequestsErrors

Number of decompression requests that had errors and could not be processed.

Cpa64U _CpaDcStats::numDecompCompleted

Decompression requests completed.

Cpa64U _CpaDcStats::numDecompCompletedErrors

Decompression requests not completed due to errors.

5.8.4 _CpaDcRqResults Struct Reference

5.8.4.1 Detailed Description

Request results data.

This structure contains the request results.

For stateful sessions the status, produced, consumed and endOfLastBlock results are per request values while the checksum value is cumulative across all requests on the session so far. In this case the checksum value is not guaranteed to be correct until the final compressed data has been processed.

For stateless sessions, an initial checksum value is passed into the stateless operation. Once the stateless operation completes, the checksum value will contain checksum produced by the operation.

5.8.4.2 Data Fields

- **CpaDcReqStatus status**
  Additional status details from accelerator.
- **Cpa32U produced**
  Octets produced by the operation.
- **Cpa32U consumed**
  Octets consumed by the operation.
- **Cpa32U checksum**
  Initial checksum passed into stateless operations.
- **CpaBoolean endOfLastBlock**
  Decompression operation has stopped at the end of the last block in a deflate stream.

5.8.4.3 Field Documentation

CpaDcReqStatus _CpaDcRqResults::status

Additional status details from accelerator.

Cpa32U _CpaDcRqResults::produced

Octets produced by the operation.

Cpa32U _CpaDcRqResults::consumed

Octets consumed by the operation.

Cpa32U _CpaDcRqResults::checksum

Initial checksum passed into stateless operations.

Will also be updated to the checksum produced by the operation

CpaBoolean _CpaDcRqResults::endOfLastBlock

Decompression operation has stopped at the end of the last block in a deflate stream.

5.9 Define Documentation

#define CPA_DC_API_VERSION_NUM_MAJOR

File: cpa_dc.h

CPA Dc Major Version Number

The CPA_DC API major version number. This number will be incremented when significant churn to the API has occurred. The combination of the major and minor number definitions represent the complete version number for this interface.

#define CPA_DC_API_VERSION_NUM_MINOR

File: cpa_dc.h

CPA DC Minor Version Number

The CPA_DC API minor version number. This number will be incremented when minor changes to the API has occurred. The combination of the major and minor number definitions represent the complete version number for this interface.

#define CPA_DC_BAD_DATA

Service specific return codesInput data in invalid.

Compression specific return codes

5.10 Typedef Documentation

typedef void* **CpaDcSessionHandle**

Compression API session handle type.

Handle used to uniquely identify a Compression API session handle. This handle is established upon registration with the API using **cpaDcInitSession()**.

typedef enum **_CpaDcFileType CpaDcFileType**

Supported file types.

This enumerated lists identified file types. Used to select Huffman trees. File types are associated with Precompiled Huffman Trees.

typedef enum **_CpaDcFlush CpaDcFlush**

Supported flush flags.

This enumerated list identifies the types of flush that can be specified for stateful and stateless cpaDcCompressData and cpaDcDecompressData functions.

typedef enum **_CpaDcHuffType CpaDcHuffType**

Supported Huffman Tree types.

This enumeration lists support for Huffman Tree types. Selecting Static Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with fixed Huffman trees".

Selecting Full Dynamic Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with dynamic Huffman codes". The headers are calculated on the data being compressed, requiring two passes.

Selecting Precompiled Huffman Trees generates blocks with RFC 1951 dynamic headers. The headers are pre-calculated and are specified by the file type.

CpaDcFileType is valid only for the deflate compression algorithm. CpaDcFileType is ignored for static or full dynamic Huffman trees.

typedef enum **_CpaDcCompType CpaDcCompType**

Supported compression types.

This enumeration lists the supported data compression algorithms. In combination with CpaDcChecksum it is used to decide on the file header and footer format.

typedef enum **_CpaDcChecksum CpaDcChecksum**

Supported checksum algorithms.

This enumeration lists the supported checksum algorithms Used to decided on file header and footer specifics.

typedef enum **_CpaDcSessionDir CpaDcSessionDir**

Supported session directions.

This enumerated list identifies the direction of a session. A session can be compress, decompress or both.

typedef enum **_CpaDcSessionState CpaDcSessionState**

Supported session state settings.

This enumerated list identifies the stateful setting of a session. A session can be either stateful or stateless.

Stateful sessions are limited to have only one in-flight message per session. This means a compress or decompress request must be complete before a new request can be started. This applies equally to sessions that are uni-directional in nature and sessions that are combined compress and decompress. Completion occurs when the synchronous function returns, or when the asynchronous call back function has completed.

typedef enum **_CpaDcCompLvl CpaDcCompLvl**

Supported compression levels.

This enumerated lists the supported compressed levels. Lower values will result in less compressibility in less time.

typedef enum **_CpaDcReqStatus CpaDcReqStatus**

Supported additional details from accelerator.

This enumeration lists the supported additional details from the accelerator. These may be useful in determining the best way to recover from a failure.

typedef enum **_CpaDcAutoSelectBest CpaDcAutoSelectBest**

Supported modes for automatically selecting the best compression type.

This enumeration lists the supported modes for automatically selecting the best huffman encoding which would lead to the best compression results.

typedef void(* **CpaDcCallbackFn**)(void *callbackTag, **CpaStatus** status)

Definition of callback function invoked for asynchronous cpaDc requests.

This is the prototype for the cpaDc compression callback functions. The callback function is registered by the application using the **cpaDcInitSession()** function call.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

callbackTag	User-supplied value to help identify request.
status	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
None

typedef struct **_CpaDcInstanceCapabilities CpaDcInstanceCapabilities**

Implementation Capabilities Structure.

This structure contains data relating to the capabilities of an implementation. The capabilities include supported compression algorithms, RFC 1951 options and whether the implementation supports both stateful and stateless compress and decompress sessions.

typedef struct **_CpaDcSessionSetupData CpaDcSessionSetupData**

Session Setup Data.

This structure contains data relating to setting up a session. The client needs to complete the information in this structure in order to setup a session.

typedef struct **_CpaDcStats CpaDcStats**

Compression Statistics Data.

This structure contains data elements corresponding to statistics. Statistics are collected on a per instance basis and include: jobs submitted and completed for both compression and decompression.

typedef struct **_CpaDcRqResults CpaDcRqResults**

Request results data.

This structure contains the request results.

For stateful sessions the status, produced, consumed and endOfLastBlock results are per request values while the checksum value is cumulative across all requests on the session so far. In this case the checksum value is not guaranteed to be correct until the final compressed data has been processed.

For stateless sessions, an initial checksum value is passed into the stateless operation. Once the stateless operation completes, the checksum value will contain checksum produced by the operation.

typedef void(* **CpaDcInstanceNotificationCbFunc**)(const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, const **CpaInstanceEvent** instanceEvent)

Callback function for instance notification support.

This is the prototype for the instance notification callback function. The callback function is passed in as a parameter to the **cpaDcInstanceSetNotificationCb** function.

Context:
This function will be executed in a context that requires that sleeping MUST NOT be permitted.

Assumptions:
None

Side-Effects:

None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCallbackTag*	Opaque value provided by user while making individual function calls.
`[in]`	*instanceEvent*	The event that will trigger this function to get invoked.

Return values:
None

Precondition:
Component has been initialized and the notification function has been set via the cpaDcInstanceSetNotificationCb function.

Postcondition:
None

Note:
None

See also:
cpaDcInstanceSetNotificationCb(),

5.11 Enumeration Type Documentation

enum **_CpaDcFileType**

Supported file types.

This enumerated lists identified file types. Used to select Huffman trees. File types are associated with Precompiled Huffman Trees.

Enumerator:

CPA_DC_FT_ASCII	ASCII File Type.
CPA_DC_FT_CSS	Cascading Style Sheet File Type.
CPA_DC_FT_HTML	HTML or XML (or similar) file type.
CPA_DC_FT_JAVA	File Java code or similar.
CPA_DC_FT_OTHER	Other file types.

enum **_CpaDcFlush**

Supported flush flags.

This enumerated list identifies the types of flush that can be specified for stateful and stateless cpaDcCompressData and cpaDcDecompressData functions.

Enumerator:

CPA_DC_FLUSH_NONE	No flush request.
CPA_DC_FLUSH_FINAL	Indicates that the input buffer contains all of the data for the compression session allowing any buffered data to be released. For Deflate, BFINAL is set in the compression header.
CPA_DC_FLUSH_SYNC	Used for stateful deflate compression to indicate that all pending output is flushed, byte aligned, to the output buffer. The session state is not reset.
CPA_DC_FLUSH_FULL	Used for deflate compression to indicate that all pending output is flushed to the output buffer and the session state is reset.

enum **_CpaDcHuffType**

Supported Huffman Tree types.

This enumeration lists support for Huffman Tree types. Selecting Static Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with fixed Huffman trees".

Selecting Full Dynamic Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with dynamic Huffman codes". The headers are calculated on the data being compressed, requiring two passes.

Selecting Precompiled Huffman Trees generates blocks with RFC 1951 dynamic headers. The headers are pre-calculated and are specified by the file type.

CpaDcFileType is valid only for the deflate compression algorithm. CpaDcFileType is ignored for static or full dynamic Huffman trees.

Enumerator:

CPA_DC_HT_STATIC	Static Huffman Trees.
CPA_DC_HT_PRECOMP	Precompiled Huffman Trees.
CPA_DC_HT_FULL_DYNAMIC	Full Dynamic Huffman Trees.

enum **_CpaDcCompType**

Supported compression types.

This enumeration lists the supported data compression algorithms. In combination with CpaDcChecksum it is used to decide on the file header and footer format.

Enumerator:

CPA_DC_LZS	LZS Compression.
CPA_DC_ELZS	Extended LZS Compression.
CPA_DC_LZSS	LZSS Compression.
CPA_DC_DEFLATE	Deflate Compression.

enum **_CpaDcChecksum**

Supported checksum algorithms.

This enumeration lists the supported checksum algorithms Used to decided on file header and footer specifics.

Enumerator:

CPA_DC_NONE	No checksums required.

CPA_DC_CRC32	application requires a CRC32 checksum
CPA_DC_ADLER32	Application requires Adler-32 checksum.

enum **_CpaDcSessionDir**

Supported session directions.

This enumerated list identifies the direction of a session. A session can be compress, decompress or both.

Enumerator:

CPA_DC_DIR_COMPRESS	Session will be used for compression.
CPA_DC_DIR_DECOMPRESS	Session will be used for decompression.
CPA_DC_DIR_COMBINED	Session will be used both both compression and decompression.

enum **_CpaDcSessionState**

Supported session state settings.

This enumerated list identifies the stateful setting of a session. A session can be either stateful or stateless.

Stateful sessions are limited to have only one in-flight message per session. This means a compress or decompress request must be complete before a new request can be started. This applies equally to sessions that are uni-directional in nature and sessions that are combined compress and decompress. Completion occurs when the synchronous function returns, or when the asynchronous call back function has completed.

Enumerator:

CPA_DC_STATEFUL	Session will be stateful, implying that state may need to be saved in some situations.
CPA_DC_STATELESS	Session will be stateless, implying no state will be stored.

enum **_CpaDcCompLvl**

Supported compression levels.

This enumerated lists the supported compressed levels. Lower values will result in less compressibility in less time.

Enumerator:

CPA_DC_L1	Compression level 1.
CPA_DC_L2	Compression level 2.
CPA_DC_L3	Compression level 3.
CPA_DC_L4	Compression level 4.
CPA_DC_L5	Compression level 5.
CPA_DC_L6	Compression level 6.
CPA_DC_L7	Compression level 7.
CPA_DC_L8	Compression level 8.
CPA_DC_L9	Compression level 9.

enum **_CpaDcReqStatus**

Supported additional details from accelerator.

This enumeration lists the supported additional details from the accelerator. These may be useful in determining the best way to recover from a failure.

Enumerator:

CPA_DC_OK	No error detected by compression slice.
CPA_DC_INVALID_BLOCK_TYPE	Invalid block type (type == 3).
CPA_DC_BAD_STORED_BLOCK_LEN	Stored block length did not match one's complement.
CPA_DC_TOO_MANY_CODES	Too many length or distance codes.
CPA_DC_INCOMPLETE_CODE_LENS	Code length codes incomplete.
CPA_DC_REPEATED_LENS	Repeated lengths with no first length.
CPA_DC_MORE_REPEAT	Repeat more than specified lengths.
CPA_DC_BAD_LITLEN_CODES	Invalid literal/length code lengths.
CPA_DC_BAD_DIST_CODES	Invalid distance code lengths.
CPA_DC_INVALID_CODE	Invalid literal/length or distance code in fixed or dynamic block.
CPA_DC_INVALID_DIST	Distance is too far back in fixed or dynamic block.
CPA_DC_OVERFLOW	Overflow detected.
CPA_DC_SOFTERR	Other non-fatal detected.
CPA_DC_FATALERR	Fatal error detected.

enum **_CpaDcAutoSelectBest**

Supported modes for automatically selecting the best compression type.

This enumeration lists the supported modes for automatically selecting the best huffman encoding which would lead to the best compression results.

Enumerator:

CPA_DC_ASB_DISABLED	Auto select best mode is disabled.
CPA_DC_ASB_STATIC_DYNAMIC	Auto select between static and dynamic compression.
CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS	Auto select between uncompressed, static and dynamic compression, using stored block deflate headers if uncompressed is selected.
CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS	Auto select between uncompressed, static and dynamic compression, using no deflate headers if uncompressed is selected.

5.12 Function Documentation

```
CpaStatus cpaDcQueryCapabilities ( CpaInstanceHandle             dcInstance,
                                   CpaDcInstanceCapabilities *   pInstanceCapabilities
                                 )
```

Retrieve Instance Capabilities.

This function is used to retrieve the capabilities matrix of an instance.

Context:
This function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle derived from discovery functions
`[in,out]`	*pInstanceCapabilities*	Pointer to a capabilities struct

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
None

Postcondition:
None

Note:
Only a synchronous version of this function is provided.

See also:
None

```
CpaStatus cpaDcInitSession ( CpaInstanceHandle          dcInstance,
                             CpaDcSessionHandle         pSessionHandle,
                             CpaDcSessionSetupData *    pSessionData,
                             CpaBufferList *            pContextBuffer,
                             CpaDcCallbackFn            callbackFn
```

)

Initialize compression decompression session.

This function is used to initialize a compression/decompression session. This function specifies a BufferList for context data. A single session can be used for both compression and decompression requests. Clients MAY register a callback function for the compression service using this function. This function returns a unique session handle each time this function is invoked. If the session has been configured with a callback function, then the order of the callbacks are guaranteed to be in the same order the compression or decompression requests were submitted for each session, so long as a single thread of execution is used for job submission.

Context:

This is a synchronous function and it cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:

No

Reentrant:

No

Thread-safe:

Yes

Parameters:

`[in]`	*dcInstance*	Instance handle derived from discovery functions.
`[in,out]`	*pSessionHandle*	Pointer to a session handle.
`[in,out]`	*pSessionData*	Pointer to a user instantiated structure containing session data.
`[in]`	*pContextBuffer*	pointer to context buffer. This is not required for stateless operations. The total size of the buffer list must be equal to or larger than the specified contextSize retrieved from the **cpaDcGetSessionSize()** function.
`[in]`	*callbackFn*	For synchronous operation this callback shall be a null pointer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:

dcInstance has been started using cpaDcStartInstance.

Postcondition:

None

Note:

Only a synchronous version of this function is provided.

This initializes opaque data structures in the session handle. Data compressed under this session will be compressed to the level specified in the pSessionData structure. Lower compression levels numbers indicate a request for faster compression at the expense of compression ratio. Higher compression level numbers indicate a request for higher compression ratios at the expense of execution time.

The session is opaque to the user application and the session handle contains job specific data.

The pointer to the ContextBuffer will be stored in session specific data if required by the implementation.

It is not permitted to have multiple outstanding asynchronous compression requests for stateful sessions. It is possible to add parallelization to compression by using multiple sessions.

The window size specified in the pSessionData must be match exactly one of the supported window sizes specified in the capabilities structure. If a bi-directional session is being initialized, then the window size must be valid for both compress and decompress.

See also:
None

CpaStatus cpaDcRemoveSession (const **CpaInstanceHandle** *dcInstance*,
CpaDcSessionHandle *pSessionHandle*
)

Compression Session Remove Function.

This function will remove a previously initialized session handle and the installed callback handler function. Removal will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the remove function at a later time. The memory for the session handle MUST not be freed until this call has completed successfully.

Context:
This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle.
`[in,out]`	*pSessionHandle*	Session handle.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.

CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
The component has been initialized via cpaDcStartInstance function.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

See also:
cpaDcInitSession()

```
CpaStatus cpaDcCompressData ( CpaInstanceHandle    dcInstance,
                              CpaDcSessionHandle   pSessionHandle,
                              CpaBufferList *      pSrcBuff,
                              CpaBufferList *      pDestBuff,
                              CpaDcRqResults *     pResults,
                              CpaDcFlush           flushFlag,
                              void *               callbackTag
                            )
```

Submit a request to compress a buffer of data.

This API consumes data from the input buffer and generates compressed data in the output buffer.

Context:
When called as an asynchronous funnction it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Target service instance.
`[in,out]`	*pSessionHandle*	Session handle.
`[in]`	*pSrcBuff*	Pointer to data buffer for compression.
`[in]`	*pDestBuff*	Pointer to buffer space for data after compression.
`[in,out]`	*pResults*	Pointer to results structure

`[in]`	*flushFlag*	Indicates the type of flush to be performed.
`[in]`	*callbackTag*	User supplied value to help correlate the callback with its associated request.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_DC_BAD_DATA	The input data was not properly formed.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:

pSessionHandle has been setup using **cpaDcInitSession()**

Postcondition:

pSessionHandle has session related state information

Note:

This function passes control to the compression service for processing

In synchronous mode the function returns the error status returned from the service. In asynchronous mode the status is returned by the callback function.

This function may be called repetitively with input until all of the input has been consumed by the compression service and all the output has been produced.

When this function returns, it may be that all of the available data in the input buffer has not been compressed. This situation will occur when there is insufficient space in the output buffer. The calling application should note the amount of data processed, and clear the output buffer and then submit the request again, with the input buffer pointer to the data that was not previously compressed.

Relationship between input buffers and results buffers.

1. Implementations of this API must not modify the individual flat buffers of the input buffer list.
2. The implementation communicates the amount of data consumed from the source buffer list via pResults->consumed arg.
3. The implementation communicates the amount of data in the destination buffer list via pResults->produced arg.

Source Buffer Setup Rules

1. The buffer list must have the correct number of flat buffers. This is specified by the numBuffers element of the CpaBufferList.
2. Each flat buffer must have a pointer to contiguous memory that has been allocated by the calling application. The number of octets to be compressed or decompressed must be stored in the dataLenInBytes element of the flat buffer.
3. It is permissible to have one or more flat buffers with a zero length data store. This function will process all flat buffers until the destination buffer is full or all source data has been processed. If a buffer has zero length, then no data will be processed from that buffer.

Source Buffer Processing Rules.

1. The buffer list is processed in index order - SrcBuff->pBuffers[0] will be completely processed before SrcBuff->pBuffers[1] begins to be processed.
2. The application must drain the destination buffers. If the source data was not completely consumed, the application must resubmit the request.
3. On return, the pResults->consumed will indicate the number of bytes consumed from the input buffers.

Destination Buffer Setup Rules

1. The destination buffer list must have storage for processed data. This implies at least one flat buffer must exist in the buffer list.
2. For each flat buffer in the buffer list, the dataLenInBytes element must be set to the size of the buffer space.
3. It is permissible to have one or more flat buffers with a zero length data store. If a buffer has zero length, then no data will be added to that buffer.

Destination Buffer Processing Rules.

1. The buffer list is processed in index order - DestBuff->pBuffers[0] will be completely processed before DestBuff->pBuffers[1] begins to be processed.
2. On return, the pResults->produced will indicate the number of bytes written to the output buffers.
3. If processing has not been completed, the application must drain the destination buffers and resubmit the request. The application must reset the dataLenInBytes for each flat buffer in the destination buffer list.

Checksum rules. If a checksum is specified in the session setup data, then:

1. For the first request for a particular data segment the checksum is initialised internally by the implementation.
2. The checksum is maintained by the implementation between calls until the flushFlag is set to CPA_DC_FLUSH_FINAL indicating the end of a particular data segment.
    a. Intermediate checksum values are returned to the application, via the CpaDcRqResults structure, in response to each request. However these checksum values are not guaranteed to the valid until the call with flushFlag set to CPA_DC_FLUSH_FINAL completes successfully.

The application should set flushFlag to CPA_DC_FLUSH_FINAL to indicate processing a particular data segment is complete. It should be noted that this function may have to be called more than once to process data after the flushFlag parameter has been set to CPA_DC_FLUSH_FINAL if the destination buffer fills. Refer to buffer processing rules.

For statelful operations, when the function is invoked with flushFlag set to CPA_DC_FLUSH_NONE or CPA_DC_FLUSH_SYNC, indicating more data is yet to come, the function may or may not retain data. When the function is invoked with flushFlag set to CPA_DC_FLUSH_FULL or CPA_DC_FLUSH_FINAL, the function will process all buffered data.

For stateless operations, CPA_DC_FLUSH_FINAL will cause the BFINAL bit to be set for deflate compression. The initial checksum for the stateless operation should be set to 0. CPA_DC_FLUSH_NONE and CPA_DC_FLUSH_SYNC should not be used for stateless operations.

It is possible to maintain checksum and length information across **cpaDcCompressData()** calls with a stateless session without maintaining the full history state that is maintained by a stateful session. In this mode of operation, an initial checksum value of 0 is passed into the first **cpaDcCompressData()** call with the flush flag set to CPA_DC_FLUSH_FULL. On subsequent calls to **cpaDcCompressData()** for this session, the checksum passed to cpaDcCompressData should be set to the checksum value produced by the previous call to **cpaDcCompressData()**. When the last block of input data is passed to

cpaDcCompressData(), the flush flag should be set to CP_DC_FLUSH_FINAL. This will cause the BFINAL bit to be set in a deflate stream. It is the responsibility of the calling application to maintain overall lengths across the stateless requests and to pass the checksum produced by one request into the next request.

Synchronous or Asynchronous operation of the API is determined by the value of the callbackFn parameter passed to **cpaDcInitSession()** when the sessionHandle was setup. If a non-NULL value was specified then the supplied callback function will be invoked asynchronously with the response of this request.

Response ordering: For each session, the implementation must maintain the order of responses. That is, if in asynchronous mode, the order of the callback functions must match the order of jobs submitted by this function. In a simple synchronous mode implementation, the practice of submitting a request and blocking on its completion ensure ordering is preserved.

This limitation does not apply if the application employs multiple threads to service a single session.

If this API is invoked asynchronous, the return code represents the success or not of asynchronously scheduling the request. The results of the operation, along with the amount of data consumed and produced become available when the callback function is invoked. As such, pResults->consumed and pResults->produced are available only when the operation is complete.

The application must not use either the source or destination buffers until the callback has completed.

See also:
None

```
CpaStatus cpaDcDecompressData ( CpaInstanceHandle   dcInstance,
                                CpaDcSessionHandle  pSessionHandle,
                                CpaBufferList *     pSrcBuff,
                                CpaBufferList *     pDestBuff,
                                CpaDcRqResults *    pResults,
                                CpaDcFlush          flushFlag,
                                void *              callbackTag
                              )
```

Submit a request to decompress a buffer of data.

This API consumes compressed data from the input buffer and generates uncompressed data in the output buffer.

Context:
When called as an asynchronous funnction it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:

Yes

Parameters:

`[in]`	*dcInstance*	Target service instance.
`[in,out]`	*pSessionHandle*	Session handle.
`[in]`	*pSrcBuff*	Pointer to data buffer for compression.
`[in]`	*pDestBuff*	Pointer to buffer space for data after decompression.
`[in,out]`	*pResults*	Pointer to results structure
`[in]`	*flushFlag*	When set to CPA_DC_FLUSH_FINAL, indicates that the input buffer contains all of the data for the compression session, allowing the function to release history data.
`[in]`	*callbackTag*	User supplied value to help correlate the callback with its associated request.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_DC_BAD_DATA	The input data was not properly formed.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:

pSessionHandle has been setup using **cpaDcInitSession()**

Postcondition:

pSessionHandle has session related state information

Note:

This function passes control to the compression service for decompression. The function returns the status from the service.

This function may be called repetitively with input until all of the input has been provided and all the output has been consumed.

This function has identical buffer processing rules as **cpaDcCompressData()**.

This function has identical checksum processing rules as **cpaDcCompressData()**.

The application should set flushFlag to CPA_DC_FLUSH_FINAL to indicate processing a particular compressed data segment is complete. It should be noted that this function may have to be called more than once to process data after flushFlag has been set if the destination buffer fills. Refer to buffer processing rules in **cpaDcCompressData()**.

Synchronous or Asynchronous operation of the API is determined by the value of the callbackFn parameter passed to **cpaDcInitSession()** when the sessionHandle was setup. If a non-NULL value was specified then the supplied callback function will be invoked asynchronously with the response of this request, along with the callbackTag specified in the function.

The same response ordering constraints identified in the cpaDcCompressData API apply to this function.

See also:

cpaDcCompressData()

CpaStatus cpaDcGenerateHeader (**CpaDcSessionHandle** *pSessionHandle*,
CpaFlatBuffer * *pDestBuff*,
Cpa32U * *count*
)

Generate compression header.

This API generates the gzip or the zlib header and stores it in the output buffer.

Context:
This function may be call from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in,out]`	*pSessionHandle*	Session handle.
`[in]`	*pDestBuff*	Pointer to data buffer where the compression header will go.
`[out]`	*count*	Pointer to counter filled in with header size.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
pSessionHandle has been setup using **cpaDcInitSession()**

Note:
This function can output a 10 byte gzip header or 2 byte zlib header to the destination buffer. The session properties are used to determine the header type. To output a header the session must have been initialized with CpaDcCompType CPA_DC_DEFLATE for any other value no header is produced. To output a gzip header the session must have been initialized with CpaDcChecksum CPA_DC_CRC32. To output a zlib header the session must have been initialized with CpaDcChecksum CPA_DC_ADLER32. For CpaDcChecksum CPA_DC_NONE no header is output.

If the compression requires a gzip header, then this header requires at a minimum the following fields, defined in RFC1952: ID1: 0x1f ID2: 0x8b CM: Compression method = 8 for deflate

The zlib header is defined in RFC1950 and this function must implement as a minimum: CM: four bit compression method - 8 is deflate with window size to 32k CINFO: four bit window size (see RFC1950 for details), 7 is 32k window FLG: defined as:

- Bits 0 - 4: check bits for CM, CINFO and FLG (see RFC1950)
- Bit 5: FDICT 0 = default, 1 is preset dictionary
- Bits 6 - 7: FLEVEL, compression level (see RFC 1950)

The counter parameter will be set to the number of bytes added to the buffer. The pData will be not be changed.

See also:
None

CpaStatus cpaDcGenerateFooter (**CpaDcSessionHandle** *pSessionHandle*, **CpaFlatBuffer** * *pDestBuff*, **CpaDcRqResults** * *pResults*)

Generate compression footer.

This API generates the footer for gzip or zlib and stores it in the output buffer.

Context:
This function may be call from any context.

Assumptions:
None

Side-Effects:
All session variables are reset

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in,out]`	*pSessionHandle*	Session handle.
`[in]`	*pDestBuff*	Pointer to data buffer where the compression footer will go.
`[in,out]`	*pResults*	Pointer to results structure filled by CpaDcCompressData. Updated with the results of this API call

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_UNSUPPORTED	Function is not supported.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
pSessionHandle has been setup using **cpaDcInitSession()** pResults structure has been filled by CpaDcCompressData().

Note:

Depending on the session variables, this function can add the alder32 footer to the zlib compressed data as defined in RFC1950. If required, it can also add the gzip footer, which is the crc32 of the uncompressed data and the length of the uncompressed data. This section is defined in RFC1952. The session variables used to determine the header type are CpaDcCompType and CpaDcChecksum, see cpaDcGenerateHeader for more details.

An artifact of invoking this function for writing the footer data is that all opaque session specific data is re-initialized. If the compression level and file types are consistent, the upper level application can continue processing compression requests using the same session handle.

The produced element of the pResults structure will be incremented by the numbers bytes added to the buffer. The pointer to the buffer will not be modified.

This function is not supported for stateless sessions.

See also:
None

CpaStatus cpaDcGetStats (**CpaInstanceHandle** *dcInstance*,
CpaDcStats * *pStatistics*
)

Retrieve statistics.

This API retrieves the current statistics for a compression instance.

Context:
This function may be call from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle.
`[out]`	*pStatistics*	Pointer to statistics structure.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:

None

Postcondition:
None

See also:
None

CpaStatus cpaDcGetNumInstances (**Cpa16U** * *pNumInstances*)

Get the number of device instances that are supported by the API implementation.

This function will get the number of device instances that are supported by an implementation of the compression API. This number is then used to determine the size of the array that must be passed to **cpaDcGetInstances()**.

Context:
This function MUST NOT be called from an interrupt context as it MAY sleep.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[out]`	*pNumInstances*	Pointer to where the number of instances will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated

See also:
cpaDcGetInstances

CpaStatus cpaDcGetInstances (**Cpa16U** *numInstances*,
dcInstances

CpaInstanceHandle *
)

Get the handles to the device instances that are supported by the API implementation.

This function will return handles to the device instances that are supported by an implementation of the compression API. These instance handles can then be used as input parameters with other compression API functions.

This function will populate an array that has been allocated by the caller. The size of this API is determined by the **cpaDcGetNumInstances()** function.

Context:
This function MUST NOT be called from an interrupt context as it MAY sleep.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*numInstances*	Size of the array.
`[out]`	*dcInstances*	Pointer to where the instance handles will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated

See also:
cpaDcGetInstances

CpaStatus cpaDcGetNumIntermediateBuffers (**CpaInstanceHandle** *instanceHandle*,
Cpa16U * *pNumBuffers*
)

Compression Component utility function to determine the number of intermediate buffers required by an implementation.

This function will determine the number of intermediate buffer lists required by an implementation for a compression instance. These buffers should then be allocated and provided when calling **cpaDcStartInstance()** to start a compression instance.

Context:
This function may sleep, and MUST NOT be called in interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in,out]`	*instanceHandle*	Handle to an instance of this API to be initialized.
`[out]`	*pNumBuffers*	When the function returns, this will specify the number of buffer lists that should be used as intermediate buffers when calling **cpaDcStartInstance()**.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Suggested course of action is to shutdown and restart.

Precondition:
None

Postcondition:
None

Note:
Note that this is a synchronous function and has no completion callback associated with it.

See also:
cpaDcStartInstance()

CpaStatus cpaDcStartInstance (**CpaInstanceHandle** *instanceHandle*,
Cpa16U *numBuffers*,
CpaBufferList ** *pIntermediateBuffers*
)

Compression Component Initialization and Start function.

This function will initialize and start the compression component. It MUST be called before any other compress function is called. This function SHOULD be called only once (either for the very first time, or after an cpaDcStopInstance call which succeeded) per instance. Subsequent calls will have no effect.

If required by an implementation, this function can be provided with instance specific intermediate buffers. The intent is to provide an instance specific location to store intermediate results during dynamic instance Huffman tree compression requests. The memory should be accessible by the compression engine. The buffers are to support deflate compression with dynamic Huffman Trees. Each buffer list should be similar in size to twice the destination buffer size passed to the compress API. The number of intermediate buffer lists may vary between implementations and so **cpaDcGetNumIntermediateBuffers()** should be called first to determine the number of intermediate buffers required by the implementation.

If not required, this parameter can be passed in as NULL.

Context:
This function may sleep, and MUST NOT be called in interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in,out]`	*instanceHandle*	Handle to an instance of this API to be initialized.
`[in]`	*numBuffers*	Number of buffer lists represented by the pIntermediateBuffers parameter. Note: **cpaDcGetNumIntermediateBuffers()** can be used to determine the number of intermediate buffers that an implementation requires.
`[in]`	*pIntermediateBuffers*	Optional pointer to Instance specific DRAM buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Suggested course of action is to shutdown and restart.

Precondition:
None

Postcondition:
None

Note:
Note that this is a synchronous function and has no completion callback associated with it.

See also:
cpaDcStopInstance() **cpaDcGetNumIntermediateBuffers()**

CpaStatus cpaDcInstanceGetInfo2 (const **CpaInstanceHandle** *instanceHandle*,
CpaInstanceInfo2 * *pInstanceInfo2*
)

Function to get information on a particular instance.

This function will provide instance specific information through a **CpaInstanceInfo2** structure.

Context:
This function will be executed in a context that requires that sleeping MUST NOT be permitted.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API to be initialized.
`[out]`	*pInstanceInfo2*	Pointer to the memory location allocated by the client into which the CpaInstanceInfo2 structure will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
The client has retrieved an instanceHandle from successive calls to **cpaDcGetNumInstances** and **cpaDcGetInstances**.

Postcondition:
None

Note:
None

See also:
cpaDcGetNumInstances, **cpaDcGetInstances**, **CpaInstanceInfo2**

CpaStatus		
cpaDcInstanceSetNotificationCb	(const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaDcInstanceNotificationCbFunc**	*pInstanceNotificationCb*,
	void *	*pCallbackTag*
	)	

Subscribe for instance notifications.

Clients of the CpaDc interface can subscribe for instance notifications by registering a **CpaDcInstanceNotificationCbFunc** function.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pInstanceNotificationCb*	Instance notification callback function pointer.
`[in]`	*pCallbackTag*	Opaque value provided by user while making individual function calls.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
Instance has been initialized.

Postcondition:
None

Note:
None

See also:
CpaDcInstanceNotificationCbFunc

```
CpaStatus cpaDcGetSessionSize ( CpaInstanceHandle          dcInstance,
                                CpaDcSessionSetupData *    pSessionData,
                                Cpa32U *                   pSessionSize,
                                Cpa32U *                   pContextSize
                              )
```

Get the size of the memory required to hold the session information.

The client of the Data Compression API is responsible for allocating sufficient memory to hold session information and the context data. This function provides a means for determining the size of the session information and the size of the context data.

Context:
No restrictions

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle.
`[in]`	*pSessionData*	Pointer to a user instantiated structure containing session data.
`[out]`	*pSessionSize*	On return, this parameter will be the size of the memory that will be required by **cpaDcInitSession()** for session data.
`[out]`	*pContextSize*	On return, this parameter will be the size of the memory that will be required for context data. Context data is save/restore data including history and any implementation specific data that is required for a save/restore operation.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

Note:
Only a synchronous version of this function is provided.

It is expected that context data is comprised of the history and any data stores that are specific to the history such as linked lists or hash tables. For stateless sessions the context size returned from this function will be zero. For stateful sessions the context size returned will depend on the session setup data.

Session data is expected to include interim checksum values, various counters and other session related data that needs to persist between invocations. For a given implementation of this API, it is safe to assume that **cpaDcGetSessionSize()** will always return the same session size and that the size will not be different for different setup data parameters. However, it should be noted that the size may change: (1) between different implementations of the API (e.g. between software and hardware implementations or between different hardware implementations) (2) between different releases of the same API implementation.

See also:
cpaDcInitSession()

CpaStatus cpaDcBufferListGetMetaSize (	const **CpaInstanceHandle**	*instanceHandle*,
	Cpa32U	*numBuffers*,
	Cpa32U *	*pSizeInBytes*

)

Function to return the size of the memory which must be allocated for the pPrivateMetaData member of CpaBufferList.

This function is used to obtain the size (in bytes) required to allocate a buffer descriptor for the pPrivateMetaData member in the CpaBufferList structure. Should the function return zero then no meta data is required for the buffer list.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API.
`[in]`	*numBuffers*	The number of pointers in the CpaBufferList. This is the maximum number of CpaFlatBuffers which may be contained in this CpaBufferList.
`[out]`	*pSizeInBytes*	Pointer to the size in bytes of memory to be allocated when the client wishes to allocate a cpaFlatBuffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

Note:
None

See also:
cpaDcGetInstances()

CpaStatus cpaDcGetStatusText (const **CpaInstanceHandle** *dcInstance*,
const **CpaStatus** *errStatus*,
Cpa8S * *pStatusText*
)

Function to return a string indicating the specific error that occurred within the system.

When a function returns any error including CPA_STATUS_SUCCESS, the client can invoke this function to get a string which describes the general error condition, and if available additional information on the specific error. The Client MUST allocate CPA_STATUS_MAX_STR_LENGTH_IN_BYTES bytes for the buffer string.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Handle to an instance of this API.
`[in]`	*errStatus*	The error condition that occurred.
`[in,out]`	*pStatusText*	Pointer to the string buffer that will be updated with the status text. The invoking application MUST allocate this buffer to be exactly CPA_STATUS_MAX_STR_LENGTH_IN_BYTES.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Note, in this scenario it is INVALID to call this function a second time.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

Note:
None

See also:
CpaStatus

CpaStatus cpaDcSetAddressTranslation (const **CpaInstanceHandle** *instanceHandle*,
CpaVirtualToPhysical *virtual2Physical*
)

Set Address Translation function.

This function is used to set the virtual to physical address translation routine for the instance. The specified routine is used by the instance to perform any required translation of a virtual address to a physical address. If the application does not invoke this function, then the instance will use its default method, such as virt2phys, for address translation.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Data Compression API instance handle.
`[in]`	*virtual2Physical*	Routine that performs virtual to physical address translation.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

See also:
None

CpaStatus cpaDcDpGetSessionSize (**CpaInstanceHandle** *dcInstance*,
CpaDcSessionSetupData * *pSessionData*,
Cpa32U * *pSessionSize*
)

Get the size of the memory required to hold the data plane session information.

The client of the Data Compression API is responsible for allocating sufficient memory to hold session information. This function provides a means for determining the size of the session information and statistics information.

Context:
No restrictions

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle.
`[in]`	*pSessionData*	Pointer to a user instantiated structure containing session data.
`[out]`	*pSessionSize*	On return, this parameter will be the size of the memory that will be required by **cpaDcInitSession()** for session data.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

Precondition:
None

Postcondition:
None

Note:
Only a synchronous version of this function is provided.

Session data is expected to include interim checksum values, various counters and other other session related data that needs to persist between invocations. For a given implementation of this API, it is safe to assume that **cpaDcDpGetSessionSize()** will always return the same session size and that the size will not be different for different setup data parameters. However, it should be noted that the size may change: (1) between different implementations of the API (e.g. between software and hardware implementations or between different hardware implementations) (2) between different releases of the same API implementation

See also:
cpaDcDpInitSession()

CpaStatus cpaDcDpRemoveSession (const **CpaInstanceHandle** *dcInstance*, **CpaDcSessionHandle** *pSessionHandle*)

Compression Data Plane Session Remove Function.

This function will remove a previously initialized session handle and the installed callback handler function. Removal will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the remove function at a later time. The memory for the session handle MUST not be freed until this call has completed successfully.

Context:

This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle.
`[in,out]`	*pSessionHandle*	Session handle.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
The component has been initialized via **cpaDcStartInstance** function.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

See also:
cpaDcDpInitSession

6 Data Compression Data Plane API

[Data Compression API]

Collaboration diagram for Data Compression Data Plane API:

6.1 Detailed Description

File: cpa_dc_dp.h

These data structures and functions specify the Data Plane API for compression and decompression operations.

This API is recommended for data plane applications, in which the cost of offload - that is, the cycles consumed by the driver in sending requests to the hardware, and processing responses - needs to be minimized. In particular, use of this API is recommended if the following constraints are acceptable to your application:

- Thread safety is not guaranteed. Each software thread should have access to its own unique instance (CpaInstanceHandle) to avoid contention.
- Polling is used, rather than interrupts (which are expensive). Implementations of this API will provide a function (not defined as part of this API) to read responses from the hardware response queue and dispatch callback functions, as specified on this API.
- Buffers and buffer lists are passed using physical addresses, to avoid virtual to physical address translation costs.
- The ability to enqueue one or more requests without submitting them to the hardware allows for certain costs to be amortized across multiple requests.
- Only asynchronous invocation is supported.
- There is no support for partial packets.
- Implementations may provide certain features as optional at build time, such as atomic counters.
- There is no support for stateful operations.
    - ♦ The "default" instance (CPA_INSTANCE_HANDLE_SINGLE) is not supported on this API. The specific handle should be obtained using the instance discovery functions (**cpaDcGetNumInstances**, **cpaDcGetInstances**).

6.2 Data Structures

- struct **_CpaDcDpOpData**

6.3 Typedefs

- typedef **_CpaDcDpOpData CpaDcDpOpData**
- typedef void(* **CpaDcDpCallbackFn**)(**CpaDcDpOpData** *pOpData)

6.4 Functions

- **CpaStatus cpaDcDpInitSession** (**CpaInstanceHandle** dcInstance, **CpaDcSessionHandle** pSessionHandle, **CpaDcSessionSetupData** *pSessionData)

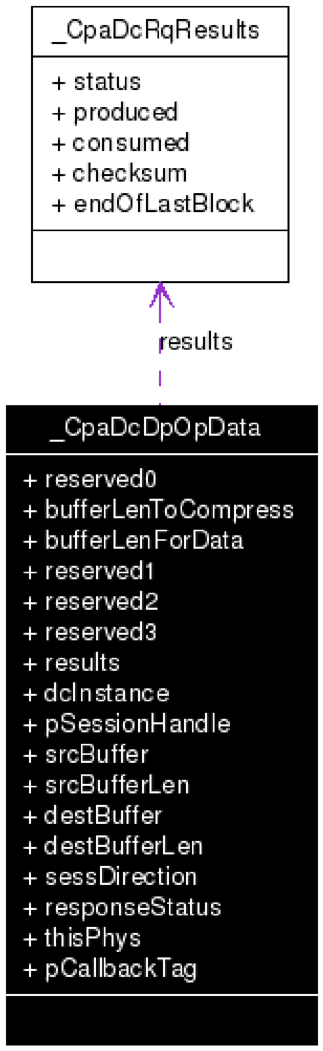
6.4 Functions

- **CpaStatus cpaDcDpRegCbFunc** (const **CpaInstanceHandle** dcInstance, const **CpaDcDpCallbackFn** pNewCb)
- **CpaStatus cpaDcDpEnqueueOp** (**CpaDcDpOpData** *pOpData, const **CpaBoolean** performOpNow)
- **CpaStatus cpaDcDpEnqueueOpBatch** (const **Cpa32U** numberRequests, **CpaDcDpOpData** *pOpData[], const **CpaBoolean** performOpNow)
- **CpaStatus cpaDcDpPerformOpNow** (**CpaInstanceHandle** dcInstance)

6.5 Data Structure Documentation

6.5.1 _CpaDcDpOpData Struct Reference

Collaboration diagram for _CpaDcDpOpData:



6.5.1.1 Detailed Description

File: cpa_dc_dp.h

Operation Data for compression data plane API.

6.5.1 _CpaDcDpOpData Struct Reference

This structure contains data relating to a request to perform compression processing on one or more data buffers.

The physical memory to which this structure points should be at least 8-byte aligned.

All reserved fields SHOULD NOT be written or read by the calling code.

See also:
cpaDcDpEnqueueOp, **cpaDcDpEnqueueOpBatch**

6.5.1.2 Data Fields

- **Cpa64U reserved0**
  Reserved for internal use.
- **Cpa32U bufferLenToCompress**
  The total size of the input data in bytes.
- **Cpa32U bufferLenForData**
  The total size of the output buffer in bytes.
- **Cpa64U reserved1**
  Reserved for internal use.
- **Cpa64U reserved2**
  Reserved for internal use.
- **Cpa64U reserved3**
  Reserved for internal use.
- **CpaDcRqResults results**
  Results of the operation.
- **CpaInstanceHandle dcInstance**
  Instance to which the request is to be enqueued.
- **CpaDcSessionHandle pSessionHandle**
  DC Session associated with the stream of requests.
- **CpaPhysicalAddr srcBuffer**
  Physical address of the source buffer on which to operate.
- **Cpa32U srcBufferLen**
  The total size of the input buffer in bytes.
- **CpaPhysicalAddr destBuffer**
  Physical address of the destination buffer on which to operate.
- **Cpa32U destBufferLen**
  The total size of the output buffer in bytes.
- **CpaDcSessionDir sessDirection**
  Session direction indicating whether session is used for compression, decompression.
- **CpaStatus responseStatus**
  Status of the operation.
- **CpaPhysicalAddr thisPhys**
  Physical address of this data structure.
- void * **pCallbackTag**
  Opaque data that will be returned to the client in the function completion callback.

6.5.1.3 Field Documentation

Cpa64U _CpaDcDpOpData::reserved0

Reserved for internal use.

Source code should not read or write this field.

Cpa32U _CpaDcDpOpData::bufferLenToCompress

The total size of the input data in bytes.

Cpa32U _CpaDcDpOpData::bufferLenForData

The total size of the output buffer in bytes.

Cpa64U _CpaDcDpOpData::reserved1

Reserved for internal use.

Source code should not read or write

Cpa64U _CpaDcDpOpData::reserved2

Reserved for internal use.

Source code should not read or write

Cpa64U _CpaDcDpOpData::reserved3

Reserved for internal use.

Source code should not read or write

CpaDcRqResults _CpaDcDpOpData::results

Results of the operation.

Contents are valid upon completion.

CpaInstanceHandle _CpaDcDpOpData::dcInstance

Instance to which the request is to be enqueued.

CpaDcSessionHandle _CpaDcDpOpData::pSessionHandle

DC Session associated with the stream of requests.

CpaPhysicalAddr _CpaDcDpOpData::srcBuffer

Physical address of the source buffer on which to operate.

This is either the location of the data, of length srcBufferLen; or, if srcBufferLen has the special value **CPA_DP_BUFLIST**, then srcBuffer contains the location where a **CpaPhysBufferList** is stored.

Cpa32U _CpaDcDpOpData::srcBufferLen

The total size of the input buffer in bytes.

If the srcBuffer is a pointer to a buffer list then this value is set to **CPA_DP_BUFLIST**

CpaPhysicalAddr _CpaDcDpOpData::destBuffer

Physical address of the destination buffer on which to operate.

This is either the location of the data, of length destBufferLen; or, if destBufferLen has the special value **CPA_DP_BUFLIST**, then destBuffer contains the location where a **CpaPhysBufferList** is stored.

Cpa32U _CpaDcDpOpData::destBufferLen

The total size of the output buffer in bytes.

If the destBuffer is a pointer to a buffer list then this value is set to **CPA_DP_BUFLIST**

CpaDcSessionDir _CpaDcDpOpData::sessDirection

Session direction indicating whether session is used for compression, decompression.

For the DP implemetnation, CPA_DC_DIR_COMBINED is not a valid selection.

CpaStatus _CpaDcDpOpData::responseStatus

Status of the operation.

Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL

CpaPhysicalAddr _CpaDcDpOpData::thisPhys

Physical address of this data structure.

void* **_CpaDcDpOpData::pCallbackTag**

Opaque data that will be returned to the client in the function completion callback.

This opaque data is not used by the implementation of the API, but is simply returned as part of the asynchronous response. It may be used to store information that might be useful when processing the response later.

6.6 Typedef Documentation

typedef struct **_CpaDcDpOpData CpaDcDpOpData**

File: cpa_dc_dp.h

Operation Data for compression data plane API.

This structure contains data relating to a request to perform compression processing on one or more data buffers.

The physical memory to which this structure points should be at least 8-byte aligned.

All reserved fields SHOULD NOT be written or read by the calling code.

See also:
cpaDcDpEnqueueOp, **cpaDcDpEnqueueOpBatch**

typedef void(* **CpaDcDpCallbackFn**)(**CpaDcDpOpData** *pOpData)

File: cpa_dc_dp.h

Definition of callback function for compression data plane API.

This is the callback function prototype. The callback function is registered by the application using the **cpaDcDpRegCbFunc** function call, and called back on completion of asycnhronous requests made via calls to **cpaDcDpEnqueueOp** or **cpaDcDpEnqueueOpBatch**.

Context:

This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
No

Parameters:

`[in]`	*pOpData*	Pointer to the **CpaDcDpOpData** object which was supplied as part of the original request.

Returns:
None

Precondition:
Instance has been initialized. Callback has been registered with **cpaDcDpRegCbFunc**.

Postcondition:
None

Note:
None

See also:
cpaDcDpRegCbFunc

6.7 Function Documentation

CpaStatus cpaDcDpInitSession (**CpaInstanceHandle** *dcInstance*,
CpaDcSessionHandle *pSessionHandle*,
CpaDcSessionSetupData * *pSessionData*
)

File: cpa_dc_dp.h

Initialize compression or decompression data plane session.

This function is used to initialize a compression/decompression session. A single session can be used for both compression and decompression requests. Clients MUST register a callback function for the compression service using this function. This function returns a unique session handle each time this function is invoked. The order of the callbacks are guaranteed to be in the same order the compression or decompression requests were submitted for each session, so long as a single thread of execution is used for job submission.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*dcInstance*	Instance handle derived from discovery functions.
`[in,out]`	*pSessionHandle*	Pointer to a session handle.
`[in,out]`	*pSessionData*	Pointer to a user instantiated structure containing session data.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
dcInstance has been started using **cpaDcStartInstance**.

Postcondition:
None

Note:
Only a synchronous version of this function is provided.

This initializes opaque data structures in the session handle. Data compressed under this session will be compressed to the level specified in the pSessionData structure. Lower compression level numbers indicate a request for faster compression at the expense of compression ratio. Higher compression level numbers indicate a request for higher compression ratios at the expense of execution time.

The session is opaque to the user application and the session handle contains job specific data.

The window size specified in the pSessionData must match exactly one of the supported window sizes specified in the capability structure. If a bi-directional session is being initialized, then the window size must be valid for both compress and decompress.

Note stateful sessions are not supported by this API.

See also:
None

CpaStatus cpaDcDpRegCbFunc	(	const **CpaInstanceHandle**	*dcInstance*,
			pNewCb

const **CpaDcDpCallbackFn**
)

File: cpa_dc_dp.h

Registration of the operation completion callback function.

This function allows a completion callback function to be registered. The registered callback function is invoked on completion of asycnhronous requests made via calls to **cpaDcDpEnqueueOp** or **cpaDcDpEnqueueOpBatch**.

Context:
This is a synchronous function and it cannot sleep. It can be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
No

Parameters:

[in]	*dcInstance*	Instance on which the callback function is to be registered.
[in]	*pNewCb*	Callback function for this instance.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
Instance has been initialized.

Postcondition:
None

Note:
None

See also:
cpaDcDpCbFunc

CpaStatus cpaDcDpEnqueueOp (**CpaDcDpOpData** * *pOpData*,
const **CpaBoolean** *performOpNow*
)

File: cpa_dp_dp.h

Enqueue a single compression or decompression request.

This function enqueues a single request to perform a compression, decompression operation.

The function is asynchronous; control is returned to the user once the request has been submitted. On completion of the request, the application may poll for responses, which will cause a callback function (registered via **cpaDcDpRegCbFunc**) to be invoked. Callbacks within a session are guaranteed to be in the same order in which they were submitted.

The following restrictions apply to the pOpData parameter:

- The memory MUST be aligned on an 8-byte boundary.
- The reserved fields of the structure MUST NOT be written to or read from.
- The structure MUST reside in physically contiguous memory.

Context:
This function will not sleep, and hence can be executed in a context that does not permit sleeping.

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
No

Parameters:

`[in]`	*pOpData*	Pointer to a structure containing the request parameters. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback, which was registered on the instance via **cpaDcDpRegCbFunc**. See the above Description for some restrictions that apply to this parameter.
`[in]`	*performOpNow*	Flag to indicate whether the operation should be performed immediately (CPA_TRUE), or simply enqueued to be performed later (CPA_FALSE). In the latter case, the request is submitted to be performed either by calling this function again with this flag set to CPA_TRUE, or by invoking the function **cpaDcDpPerformOpNow**.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
The session identified by pOpData->pSessionHandle was setup using **cpaDcDpInitSession**. The instance identified by pOpData->dcInstance has had a callback function registered via **cpaDcDpRegCbFunc**.

Postcondition:
None

Note:
A callback of type **CpaDcDpCallbackFn** is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code.

See also:
cpaDcDpPerformOpNow

```
CpaStatus cpaDcDpEnqueueOpBatch ( const Cpa32U      numberRequests,
                                  CpaDcDpOpData *   pOpData[],
                                  const CpaBoolean  performOpNow
                                )
```

File: cpa_dc_dp.h

Enqueue multiple requests to the compression data plane API.

This function enqueues multiple requests to perform compression or decompression operations.

The function is asynchronous; control is returned to the user once the request has been submitted. On completion of the request, the application may poll for responses, which will cause a callback function (registered via **cpaDcDpRegCbFunc**) to be invoked. Separate callbacks will be invoked for each request. Callbacks within a session and at the same priority are guaranteed to be in the same order in which they were submitted.

The following restrictions apply to each element of the pOpData array:

- The memory MUST be aligned on an 8-byte boundary.
- The reserved fields of the structure MUST be set to zero.
- The structure MUST reside in physically contiguous memory.

Context:
This function will not sleep, and hence can be executed in a context that does not permit sleeping.

Assumptions:
Client MUST allocate the request parameters to 8 byte alignment. Reserved elements of the CpaDcDpOpData structure MUST not used The CpaDcDpOpData structure MUST reside in physically contiguous memory.

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
No

Parameters:

`[in]`	*numberRequests*	The number of requests in the array of CpaDcDpOpData structures.
`[in]`	*pOpData*	

		An array of pointers to CpaDcDpOpData structures. Each CpaDcDpOpData structure contains the request parameters for that request. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback, which was registered on the instance via **cpaDcDpRegCbFunc**. See the above Description for some restrictions that apply to this parameter.
`[in]`	*performOpNow*	Flag to indicate whether the operation should be performed immediately (CPA_TRUE), or simply enqueued to be performed later (CPA_FALSE). In the latter case, the request is submitted to be performed either by calling this function again with this flag set to CPA_TRUE, or by invoking the function **cpaDcDpPerformOpNow**.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:

The session identified by pOpData[i]->pSessionHandle was setup using **cpaDcDpInitSession**. The instance identified by pOpData[i]->dcInstance has had a callback function registered via **cpaDcDpRegCbFunc**.

Postcondition:

None

Note:

Multiple callbacks of type **CpaDcDpCallbackFn** are generated in response to this function call (one per request). Any errors generated during processing are reported as part of the callback status code.

See also:

cpaDcDpEnqueueOp

CpaStatus cpaDcDpPerformOpNow (**CpaInstanceHandle** *dcInstance*)

File: cpa_dp_dp.h

Submit any previously enqueued requests to be performed now on the compression data plane API.

This function triggers processing of previously enqueed requests on the referenced instance.

Context:

Will not sleep. It can be executed in a context that does not permit sleeping.

Side-Effects:

None

Blocking:

No

Reentrant:

No

Thread-safe:
No

Parameters:
`[in]` *dcInstance* Instance to which the requests will be submitted.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
The component has been initialized via **cpaDcStartInstance** function. A compression session has been previously setup using the **cpaDcDpInitSession** function call.

Postcondition:
None

See also:
cpaDcDpEnqueueOp, **cpaDcDpEnqueueOpBatch**