Intel® QuickAssist Technology Compression API Reference

Automatically generated from sources, Thu Apr 7 2022.

Reference Number: 330686-014

0.1 Revision History

Date	Revision	Description
April 2022	014	Compression API v3.1. Removing deprecated types, CpaDcFileType. Added support for LZ4 and LZ4s. Added support for sessionless (Ns) functions.
July 2021	013	Compression API v2.7. Added CPA_DC_ASB_ENABLED. Added a flag to notify of an uncompressed block in CpaDcRqResults.
Nov 2020	012	Compression API v2.6. Added support for integrity CRCs.
Sept 2020	011	Compression API v2.5. Added cpaDcDeflateCompressBound()
June 2020	010	Compression API v2.4. Added cpaDcUpdateSession()
Jan 2019	009	Compression API v2.3. Added chaining support for compression & hash operations.
April 2018	008	Compression API v2.2. Added support for compressAndVerifyAndRecover
Feb 2018	007	Adding support for Compress and Verify strict mode
June 2016	006	Compression API v2.1. Added support for Compress and Verify Added support for Batch and Pack Compression
October 2015	005	Compression API v2.0. Added new error codes to CpaDcReqStatus in cpa_dc.h
Sept 2015	004	Compression API v1.6. Added CPA_STATUS_UNSUPPORTED as a return status Deprecating use of fileType and deflateWindowSize fields from CpaDcSessionSetupData in cpa_dc.h Clarifying documentation for srcBufferLen, bufferLenToCompress, destBufferLen and bufferLenForData in CpaDcDpOpData in cpa_dc_dp.h
August 2015	003	Compression API v1.5. Adding reportParityError to the DC instance capabilities.
October 2014	002	Adding cpaDcResetSession API.
June 2014	001	First "public" version of the document. Based on "Intel Confidential" document number 410926-1.3.

Chapter 1

Deprecated List

Global CPA_DEPRECATED

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.

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

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 CpaOperationalState.

Global CPA_DEPRECATED

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

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

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

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

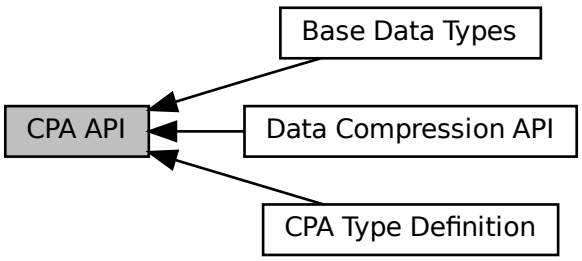
Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 CPA API

Collaboration diagram for CPA API:



Modules

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

Functions

- CpaStatus cpaGetNumInstances (const CpaAccelerationServiceType accelerationServiceType, Cpa16U *pNumInstances)
- CpaStatus cpaGetInstances (const CpaAccelerationServiceType accelerationServiceType, Cpa16U numInstances, CpaInstanceHandle *cpaInstances)

4.1.1 Detailed Description

File: cpa.h

Description:

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.

4.1.2 Function Documentation

4.1.2.1 cpaGetNumInstances()

```
CpaStatus cpaGetNumInstances (
            const CpaAccelerationServiceType accelerationServiceType,
            Cpa16U * pNumInstances )
```

File: cpa.h

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

Description:

This function will get the number of instances that are supported for the required Acceleration Service by an implementation of the CPA API. This number is then used to determine the size of the array that must be passed to cpaGetInstances().

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	*accelerationServiceType*	Acceleration Service required
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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

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

See also

cpaGetInstances

4.1.2.2 cpaGetInstances()

```
CpaStatus cpaGetInstances (
            const CpaAccelerationServiceType accelerationServiceType,
            Cpa16U numInstances,
            CpaInstanceHandle * cpaInstances )
```

File: cpa.h

Get the handles to the required Acceleration Service instances that are supported by the API implementation.

Description:

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

This function will populate an array that has been allocated by the caller. The size of this array will have been determined by the cpaGetNumInstances() 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`	*accelerationServiceType*	Acceleration Service requested
`in`	*numInstances*	Size of the array. If the value is greater than the number of instances supported, then an error (CPA_STATUS_INVALID_PARAM) is returned.
`in,out`	*cpaInstances*	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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

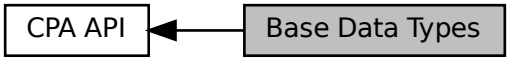
This function operates in a synchronous manner and no asynchronous callback will be generated

See also

cpaGetNumInstances

4.2 Base Data Types

Collaboration diagram for Base Data Types:



Data Structures

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

Macros

- #define CPA_INSTANCE_HANDLE_SINGLE
- #define CPA_DP_BUFLIST
- #define CPA_STATUS_SUCCESS
- #define CPA_STATUS_FAIL
- #define CPA_STATUS_RETRY
- #define CPA_STATUS_RESOURCE
- #define CPA_STATUS_INVALID_PARAM
- #define CPA_STATUS_FATAL
- #define CPA_STATUS_UNSUPPORTED
- #define CPA_STATUS_RESTARTING
- #define CPA_STATUS_MAX_STR_LENGTH_IN_BYTES
- #define CPA_STATUS_STR_SUCCESS
- #define CPA_STATUS_STR_FAIL
- #define CPA_STATUS_STR_RETRY
- #define CPA_STATUS_STR_RESOURCE
- #define CPA_STATUS_STR_INVALID_PARAM
- #define CPA_STATUS_STR_FATAL
- #define CPA_STATUS_STR_UNSUPPORTED
- #define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES
- #define CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES
- #define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES

Typedefs

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

Enumerations

- enum _CpaInstanceType
- enum _CpaAccelerationServiceType
- enum _CpaInstanceState
- enum _CpaOperationalState
- enum _CpaInstanceEvent

4.2.1 Detailed Description

File: cpa.h

Description:

The base data types for the Intel CPA API.

4.2.2 Macro Definition Documentation

4.2.2.1 CPA_INSTANCE_HANDLE_SINGLE

```
#define CPA_INSTANCE_HANDLE_SINGLE
```

Default instantiation handle value where there is only a single instance

Description:

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

Definition at line 70 of file cpa.h.

4.2.2.2 CPA_DP_BUFLIST

```
#define CPA_DP_BUFLIST
```

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.

Definition at line 243 of file cpa.h.

4.2.2.3 CPA_STATUS_SUCCESS

```
#define CPA_STATUS_SUCCESS
```

Success status value.

Definition at line 258 of file cpa.h.

4.2.2.4 CPA_STATUS_FAIL

```
#define CPA_STATUS_FAIL
```

Fail status value.

Definition at line 262 of file cpa.h.

4.2.2.5 CPA_STATUS_RETRY

```
#define CPA_STATUS_RETRY
```

Retry status value.

Definition at line 266 of file cpa.h.

4.2.2.6 CPA_STATUS_RESOURCE

```
#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.

Definition at line 270 of file cpa.h.

4.2.2.7 CPA_STATUS_INVALID_PARAM

```
#define CPA_STATUS_INVALID_PARAM
```

Invalid parameter has been passed in.

Definition at line 276 of file cpa.h.

4.2.2.8 CPA_STATUS_FATAL

```
#define CPA_STATUS_FATAL
```

A serious error has occurred. Recommended course of action is to shutdown and restart the component.

Definition at line 280 of file cpa.h.

4.2.2.9 CPA_STATUS_UNSUPPORTED

```
#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.

Definition at line 285 of file cpa.h.

4.2.2.10 CPA_STATUS_RESTARTING

```
#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.

Definition at line 291 of file cpa.h.

4.2.2.11 CPA_STATUS_MAX_STR_LENGTH_IN_BYTES

```
#define CPA_STATUS_MAX_STR_LENGTH_IN_BYTES
```

API status string type definition

Description:

> 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.

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

Definition at line 309 of file cpa.h.

4.2.2.12 CPA_STATUS_STR_SUCCESS

```
#define CPA_STATUS_STR_SUCCESS
```

Status string for CPA_STATUS_SUCCESS.

Definition at line 315 of file cpa.h.

4.2.2.13 CPA_STATUS_STR_FAIL

```
#define CPA_STATUS_STR_FAIL
```

Status string for CPA_STATUS_FAIL.

Definition at line 319 of file cpa.h.

4.2.2.14 CPA_STATUS_STR_RETRY

```
#define CPA_STATUS_STR_RETRY
```

Status string for CPA_STATUS_RETRY.

Definition at line 323 of file cpa.h.

4.2.2.15 CPA_STATUS_STR_RESOURCE

```
#define CPA_STATUS_STR_RESOURCE
```

Status string for CPA_STATUS_RESOURCE.

Definition at line 327 of file cpa.h.

4.2.2.16 CPA_STATUS_STR_INVALID_PARAM

```
#define CPA_STATUS_STR_INVALID_PARAM
```

Status string for CPA_STATUS_INVALID_PARAM.

Definition at line 331 of file cpa.h.

4.2.2.17 CPA_STATUS_STR_FATAL

```
#define CPA_STATUS_STR_FATAL
```

Status string for CPA_STATUS_FATAL.

Definition at line 335 of file cpa.h.

4.2.2.18 CPA_STATUS_STR_UNSUPPORTED

```
#define CPA_STATUS_STR_UNSUPPORTED
```

Status string for CPA_STATUS_UNSUPPORTED.

Definition at line 339 of file cpa.h.

4.2.2.19 CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES

```
#define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES
```

Maximum instance info name string length in bytes

Definition at line 437 of file cpa.h.

4.2.2.20 CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES

```
#define CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES
```

Maximum instance info id string length in bytes

Definition at line 441 of file cpa.h.

4.2.2.21 CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES

```
#define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES
```

Maximum instance info version string length in bytes

Definition at line 445 of file cpa.h.

4.2.3 Typedef Documentation

4.2.3.1 CpaInstanceHandle

```
typedef void* CpaInstanceHandle
```

Instance handle type.

Description:

Handle used to uniquely identify an instance.

Note

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

Definition at line 59 of file cpa.h.

4.2.3.2 CpaPhysicalAddr

```
typedef Cpa64U CpaPhysicalAddr
```

Physical memory address.

Description:

Type for physical memory addresses.

Definition at line 79 of file cpa.h.

4.2.3.3 CpaVirtualToPhysical

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

Virtual to physical address conversion routine.

Description:

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

Definition at line 115 of file cpa.h.

4.2.3.4 CpaFlatBuffer

```
typedef struct _CpaFlatBuffer CpaFlatBuffer
```

Flat buffer structure containing a pointer and length member.

Description:

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.

4.2.3.5 CpaBufferList

```
typedef struct _CpaBufferList CpaBufferList
```

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

Description:

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.

4.2.3.6 CpaPhysFlatBuffer

```
typedef struct _CpaPhysFlatBuffer CpaPhysFlatBuffer
```

Flat buffer structure with physical address.

Description:

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

4.2.3.7 CpaPhysBufferList

```
typedef struct _CpaPhysBufferList CpaPhysBufferList
```

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

Description:

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.

4.2.3.8 CpaStatus

```
typedef Cpa32S CpaStatus
```

API status value type definition

Description:

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.

Definition at line 256 of file cpa.h.

4.2.3.9 CPA_DEPRECATED

```
typedef struct _CpaInstanceInfo CPA_DEPRECATED
```

Instance Types

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

Description:

Enumeration of the different instance types.

Instance State

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

Description:

Enumeration of the different instance states that are possible.

Instance Info Structure

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

Description:

Structure that contains the information to describe the instance.

4.2.3.10 CpaAccelerationServiceType

```
typedef enum _CpaAccelerationServiceType CpaAccelerationServiceType
```

Service Type

Description:

Enumeration of the different service types.

4.2.3.11 CpaOperationalState

```
typedef enum _CpaOperationalState CpaOperationalState
```

Instance operational state

Description:

Enumeration of the different operational states that are possible.

4.2.3.12 CpaPhysicalInstanceId

```
typedef struct _CpaPhysicalInstanceId CpaPhysicalInstanceId
```

Physical Instance ID

Description:

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.

4.2.3.13 CpaInstanceInfo2

```
typedef struct _CpaInstanceInfo2 CpaInstanceInfo2
```

Instance Info Structure, version 2

Description:

Structure that contains the information to describe the instance.

4.2.3.14 CpaInstanceEvent

```
typedef enum _CpaInstanceEvent CpaInstanceEvent
```

Instance Events

Description:

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

4.2.4 Enumeration Type Documentation

4.2.4.1 _CpaInstanceType

```
enum _CpaInstanceType
```

Instance Types

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

Description:

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

Definition at line 357 of file cpa.h.

4.2.4.2 _CpaAccelerationServiceType

```
enum _CpaAccelerationServiceType
```

Service Type

Description:

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
CPA_ACC_SVC_TYPE_CRYPTO_ASYM	Cryptography - Asymmetric service
CPA_ACC_SVC_TYPE_CRYPTO_SYM	Cryptography - Symmetric service

Definition at line 379 of file cpa.h.

4.2.4.3 _CpaInstanceState

`enum _CpaInstanceState`

Instance State

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

Description:

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.

Definition at line 412 of file cpa.h.

4.2.4.4 _CpaOperationalState

`enum _CpaOperationalState`

Instance operational state

Description:

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.

Definition at line 428 of file cpa.h.

4.2.4.5 _CpaInstanceEvent

`enum _CpaInstanceEvent`

Instance Events

Description:

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

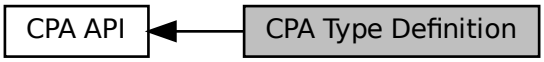
Enumerator

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.
CPA_INSTANCE_EVENT_FATAL_ERROR	Event type that triggers the registered instance notification callback function when an error has been detected that requires the device to be reset. This event will be sent by all instances using the device, both on the host and guests.

Definition at line 622 of file cpa.h.

4.3 CPA Type Definition

Collaboration diagram for CPA Type Definition:



Macros

- #define NULL
- #define CPA_BITMAP(name, sizeInBits)
- #define CPA_BITMAP_BIT_TEST(bitmask, bit)
- #define CPA_BITMAP_BIT_SET(bitmask, bit)
- #define CPA_BITMAP_BIT_CLEAR(bitmask, bit)
- #define CPA_DEPRECATED

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

Enumerations

- enum _CpaBoolean

4.3.1 Detailed Description

File: cpa_types.h

Description:

This is the CPA Type Definitions.

4.3.2 Macro Definition Documentation

4.3.2.1 NULL

```
#define NULL
```

File: cpa_types.h

NULL definition.

Definition at line 119 of file cpa_types.h.

4.3.2.2 CPA_BITMAP

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

Declare a bitmap of specified size (in bits).

Description:

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.

Definition at line 158 of file cpa_types.h.

4.3.2.3 CPA_BITMAP_BIT_TEST

```
#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).

Definition at line 161 of file cpa_types.h.

4.3.2.4 CPA_BITMAP_BIT_SET

```
#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.

Definition at line 171 of file cpa_types.h.

4.3.2.5 CPA_BITMAP_BIT_CLEAR

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

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

Definition at line 181 of file cpa_types.h.

4.3.2.6 CPA_DEPRECATED

```
typedef struct _CpaInstanceInfo CPA_DEPRECATED
```

Description:

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

Instance State

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

Description:

Enumeration of the different instance states that are possible.

Instance Info Structure

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

Description:

Structure that contains the information to describe the instance.

Definition at line 219 of file cpa_types.h.

4.3.3 Typedef Documentation

4.3.3.1 Cpa8U

```
typedef uint8_t Cpa8U
```

File: cpa_types.h

Unsigned byte base type.

Definition at line 74 of file cpa_types.h.

4.3.3.2 Cpa8S

```
typedef int8_t Cpa8S
```

File: cpa_types.h

Signed byte base type.

Definition at line 79 of file cpa_types.h.

4.3.3.3 Cpa16U

```
typedef uint16_t Cpa16U
```

File: cpa_types.h

Unsigned double-byte base type.

Definition at line 84 of file cpa_types.h.

4.3.3.4 Cpa16S

```
typedef int16_t Cpa16S
```

File: cpa_types.h

Signed double-byte base type.

Definition at line 89 of file cpa_types.h.

4.3.3.5 Cpa32U

```
typedef uint32_t Cpa32U
```

File: cpa_types.h

Unsigned quad-byte base type.

Definition at line 94 of file cpa_types.h.

4.3.3.6 Cpa32S

```
typedef int32_t Cpa32S
```

File: cpa_types.h

Signed quad-byte base type.

Definition at line 99 of file cpa_types.h.

4.3.3.7 Cpa64U

```
typedef uint64_t Cpa64U
```

File: cpa_types.h

Unsigned double-quad-byte base type.

Definition at line 104 of file cpa_types.h.

4.3.3.8 Cpa64S

```
typedef int64_t Cpa64S
```

File: cpa_types.h

Signed double-quad-byte base type.

Definition at line 109 of file cpa_types.h.

4.3.3.9 CpaBoolean

```
typedef enum _CpaBoolean CpaBoolean
```

Boolean type.

Description:

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

4.3.4 Enumeration Type Documentation

4.3.4.1 _CpaBoolean

```
enum _CpaBoolean
```

Boolean type.

Description:

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

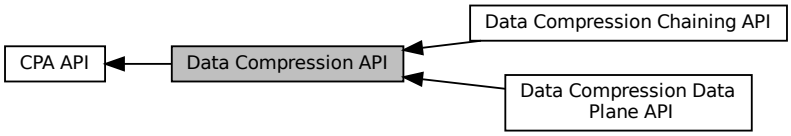
Enumerator

CPA_FALSE	False value
CPA_TRUE	True value

Definition at line 136 of file cpa_types.h.

4.4 Data Compression API

Collaboration diagram for Data Compression API:



Modules

- Data Compression Data Plane API
- Data Compression Chaining API

Data Structures

- struct _CpaDcInstanceCapabilities
- struct _CpaDcSessionSetupData
- struct _CpaDcSessionUpdateData
- struct _CpaDcStats
- struct _CpaDcRqResults
- struct _CpaIntegrityCrc
- struct _CpaIntegrityCrc64b
- struct _CpaCrcData
- struct _CpaDcSkipData
- struct _CpaDcOpData

Macros

- #define CPA_DC_API_VERSION_NUM_MAJOR
- #define CPA_DC_API_VERSION_NUM_MINOR
- #define CPA_DC_API_VERSION_AT_LEAST(major, minor)
- #define CPA_DC_API_VERSION_LESS_THAN(major, minor)
- #define CPA_DC_CHAIN_CAP_BITMAP_SIZE
- #define CPA_DC_BAD_DATA

Typedefs

- typedef void * CpaDcSessionHandle
- typedef enum _CpaDcFlush CpaDcFlush
- typedef enum _CpaDcHuffType CpaDcHuffType
- typedef enum _CpaDcCompType CpaDcCompType
- typedef enum _CpaDcCompWindowSize CpaDcCompWindowSize
- typedef enum _CpaDcCompMinMatch CpaDcCompMinMatch
- typedef enum _CpaDcCompLZ4BlockMaxSize CpaDcCompLZ4BlockMaxSize
- typedef enum _CpaDcChecksum CpaDcChecksum
- typedef enum _CpaDcSessionDir CpaDcSessionDir
- typedef enum _CpaDcSessionState CpaDcSessionState
- typedef enum _CpaDcCompLvl CpaDcCompLvl
- typedef enum _CpaDcReqStatus CpaDcReqStatus
- typedef enum _CpaDcAutoSelectBest CpaDcAutoSelectBest
- typedef enum _CpaDcSkipMode CpaDcSkipMode
- typedef void(* CpaDcCallbackFn) (void *callbackTag, CpaStatus status)
- typedef struct _CpaDcInstanceCapabilities CpaDcInstanceCapabilities
- typedef struct _CpaDcSessionSetupData CpaDcSessionSetupData
- typedef struct _CpaDcSessionUpdateData CpaDcSessionUpdateData
- typedef struct _CpaDcStats CpaDcStats
- typedef struct _CpaDcRqResults CpaDcRqResults
- typedef enum _CpaDcIntegrityCrcSize CpaDcIntegrityCrcSize
- typedef struct _CpaIntegrityCrc CpaIntegrityCrc
- typedef struct _CpaIntegrityCrc64b CpaIntegrityCrc64b
- typedef struct _CpaCrcData CpaCrcData
- typedef struct _CpaDcSkipData CpaDcSkipData
- typedef struct _CpaDcOpData CpaDcOpData
- typedef void(* CpaDcInstanceNotificationCbFunc) (const CpaInstanceHandle instanceHandle, void *pCallbackTag, const CpaInstanceEvent instanceEvent)

Enumerations

- enum _CpaDcFlush
- enum _CpaDcHuffType
- enum _CpaDcCompType
- enum _CpaDcCompWindowSize
- enum _CpaDcCompMinMatch
- enum _CpaDcCompLZ4BlockMaxSize
- enum _CpaDcChecksum
- enum _CpaDcSessionDir
- enum _CpaDcSessionState
- enum _CpaDcCompLvl
- enum _CpaDcReqStatus
- enum _CpaDcAutoSelectBest
- enum _CpaDcSkipMode
- enum _CpaDcIntegrityCrcSize

Functions

- CpaStatus cpaDcQueryCapabilities (CpaInstanceHandle dcInstance, CpaDcInstanceCapabilities *pInstanceCapabilities)
- CpaStatus cpaDcInitSession (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaDcSessionSetupData *pSessionData, CpaBufferList *pContextBuffer, CpaDcCallbackFn callbackFn)
- CpaStatus cpaDcResetSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)
- CpaStatus cpaDcResetXXHashState (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)
- CpaStatus cpaDcUpdateSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaDcSessionUpdateData *pSessionUpdateData)
- CpaStatus cpaDcRemoveSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)
- CpaStatus cpaDcDeflateCompressBound (const CpaInstanceHandle dcInstance, CpaDcHuffType huffType, Cpa32U inputSize, Cpa32U *outputSize)
- CpaStatus cpaDcLZ4CompressBound (const CpaInstanceHandle dcInstance, Cpa32U inputSize, Cpa32U *outputSize)
- CpaStatus cpaDcLZ4SCompressBound (const CpaInstanceHandle dcInstance, Cpa32U inputSize, Cpa32U *outputSize)
- CpaStatus cpaDcCompressData (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcRqResults *pResults, CpaDcFlush flushFlag, void *callbackTag)
- CpaStatus cpaDcCompressData2 (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcOpData *pOpData, CpaDcRqResults *pResults, void *callbackTag)
- CpaStatus cpaDcNsCompressData (CpaInstanceHandle dcInstance, CpaDcNsSetupData *pSetupData, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcOpData *pOpData, CpaDcRqResults *pResults, CpaDcCallbackFn callbackFn, void *callbackTag)
- CpaStatus cpaDcDecompressData (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcRqResults *pResults, CpaDcFlush flushFlag, void *callbackTag)
- CpaStatus cpaDcDecompressData2 (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcOpData *pOpData, CpaDcRqResults *pResults, void *callbackTag)
- CpaStatus cpaDcNsDecompressData (CpaInstanceHandle dcInstance, CpaDcNsSetupData *pSetupData, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcOpData *pOpData, CpaDcRqResults *pResults, CpaDcCallbackFn callbackFn, void *callbackTag)
- CpaStatus cpaDcGenerateHeader (CpaDcSessionHandle pSessionHandle, CpaFlatBuffer *pDestBuff, Cpa32U *count)
- CpaStatus cpaDcGenerateFooter (CpaDcSessionHandle pSessionHandle, CpaFlatBuffer *pDestBuff, CpaDcRqResults *pResults)
- CpaStatus cpaDcNsGenerateHeader (CpaDcNsSetupData *pSetupData, CpaFlatBuffer *pDestBuff, Cpa32U *count)
- CpaStatus cpaDcNsGenerateFooter (CpaDcNsSetupData *pSetupData, Cpa64U totalLength, CpaFlatBuffer *pDestBuff, CpaDcRqResults *pResults)
- CpaStatus cpaDcGetStats (CpaInstanceHandle dcInstance, CpaDcStats *pStatistics)
- CpaStatus cpaDcGetNumInstances (Cpa16U *pNumInstances)
- CpaStatus cpaDcGetInstances (Cpa16U numInstances, CpaInstanceHandle *dcInstances)
- CpaStatus cpaDcGetNumIntermediateBuffers (CpaInstanceHandle instanceHandle, Cpa16U *pNumBuffers)
- CpaStatus cpaDcStartInstance (CpaInstanceHandle instanceHandle, Cpa16U numBuffers, CpaBufferList **pIntermediateBuffers)
- CpaStatus cpaDcStopInstance (CpaInstanceHandle instanceHandle)
- CpaStatus cpaDcInstanceGetInfo2 (const CpaInstanceHandle instanceHandle, CpaInstanceInfo2 *pInstanceInfo2)

- CpaStatus cpaDcInstanceSetNotificationCb (const CpaInstanceHandle instanceHandle, const CpaDcInstanceNotificationCbFunc pInstanceNotificationCb, void *pCallbackTag)
- CpaStatus cpaDcGetSessionSize (CpaInstanceHandle dcInstance, CpaDcSessionSetupData *pSessionData, Cpa32U *pSessionSize, Cpa32U *pContextSize)
- CpaStatus cpaDcBufferListGetMetaSize (const CpaInstanceHandle instanceHandle, Cpa32U numBuffers, Cpa32U *pSizeInBytes)
- CpaStatus cpaDcGetStatusText (const CpaInstanceHandle dcInstance, const CpaStatus errStatus, Cpa8S *pStatusText)
- CpaStatus cpaDcSetAddressTranslation (const CpaInstanceHandle instanceHandle, CpaVirtualToPhysical virtual2Physical)
- CpaStatus cpaDcDpGetSessionSize (CpaInstanceHandle dcInstance, CpaDcSessionSetupData *pSessionData, Cpa32U *pSessionSize)
- CpaStatus cpaDcDpUpdateSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaDcSessionUpdateData *pSessionUpdateData)
- CpaStatus cpaDcDpRemoveSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)

4.4.1 Detailed Description

File: cpa_dc.h

Description:

These functions specify the API for Data Compression operations.

The Data Compression API has the following: 1) Session based API functions These functions require a session to be created before performing any DC operations. Subsequent DC API functions make use of the returned Session Handle within their structures or function prototypes. 2) Session-less or No-Session (Ns) based API functions. These functions do not require a session to be initialized before performing DC operations. They are "one-shot" API function calls that submit DC requests directly using the supplied parameters.

Remarks

4.4.2 Macro Definition Documentation

4.4.2.1 CPA_DC_API_VERSION_NUM_MAJOR

```
#define CPA_DC_API_VERSION_NUM_MAJOR
```

CPA Dc Major Version Number

Description:

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.

Definition at line 61 of file cpa_dc.h.

4.4.2.2 CPA_DC_API_VERSION_NUM_MINOR

```
#define CPA_DC_API_VERSION_NUM_MINOR
```

CPA DC Minor Version Number

Description:

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.

Definition at line 74 of file cpa_dc.h.

4.4.2.3 CPA_DC_API_VERSION_AT_LEAST

```
#define CPA_DC_API_VERSION_AT_LEAST(
              major,
              minor )
```

File: cpa_dc.h

CPA DC API version at least

Description:

The minimal supported CPA_DC API version. Allow to check if the API version is equal or above some version to avoid compilation issues with an older API version.

Definition at line 87 of file cpa_dc.h.

4.4.2.4 CPA_DC_API_VERSION_LESS_THAN

```
#define CPA_DC_API_VERSION_LESS_THAN(
              major,
              minor )
```

File: cpa_dc.h

CPA DC API version less than

Description:

The maximum supported CPA_DC API version. Allow to check if the API version is below some version to avoid compilation issues with a newer API version.

Definition at line 103 of file cpa_dc.h.

4.4.2.5 CPA_DC_CHAIN_CAP_BITMAP_SIZE

```
#define CPA_DC_CHAIN_CAP_BITMAP_SIZE
```

Size of bitmap needed for compression chaining capabilities.

Description:

Defines the number of bits in the bitmap to represent supported chaining capabilities dcChainCapInfo. Should be set to at least one greater than the largest value in the enumerated type CpaDcChainOperations, so that the value of the enum constant can also be used as the bit position in the bitmap.

A larger value was chosen to allow for extensibility without the need to change the size of the bitmap (to ease backwards compatibility in future versions of the API).

Definition at line 125 of file cpa_dc.h.

4.4.2.6 CPA_DC_BAD_DATA

```
#define CPA_DC_BAD_DATA
```

Service specific return codes

Description:

Compression specific return codes Input data in invalid

Definition at line 535 of file cpa_dc.h.

4.4.3 Typedef Documentation

4.4.3.1 CpaDcSessionHandle

```
typedef void* CpaDcSessionHandle
```

Compression API session handle type

Description:

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

Definition at line 140 of file cpa_dc.h.

4.4.3.2 CpaDcFlush

```
typedef enum _CpaDcFlush CpaDcFlush
```

Supported flush flags

Description:

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

4.4.3.3 CpaDcHuffType

```
typedef enum _CpaDcHuffType CpaDcHuffType
```

Supported Huffman Tree types

Description:

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.

4.4.3.4 CpaDcCompType

```
typedef enum _CpaDcCompType CpaDcCompType
```

Supported compression types

Description:

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

4.4.3.5 CpaDcCompWindowSize

```
typedef enum _CpaDcCompWindowSize CpaDcCompWindowSize
```

Support for defined algorithm window sizes

Description:

This enumerated list defines the valid window sizes that can be used with the supported algorithms

4.4.3.6 CpaDcCompMinMatch

```
typedef enum _CpaDcCompMinMatch CpaDcCompMinMatch
```

Min match size in bytes

Description:

This is the min match size that will be used for the search algorithm. It is only configurable for LZ4S.

4.4.3.7 CpaDcCompLZ4BlockMaxSize

```
typedef enum _CpaDcCompLZ4BlockMaxSize CpaDcCompLZ4BlockMaxSize
```

Maximum LZ4 output block size

Description:

Maximum LZ4 output block size

4.4.3.8 CpaDcChecksum

```
typedef enum _CpaDcChecksum CpaDcChecksum
```

Supported checksum algorithms

Description:

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

4.4.3.9 CpaDcSessionDir

```
typedef enum _CpaDcSessionDir CpaDcSessionDir
```

Supported session directions

Description:

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

4.4.3.10 CpaDcSessionState

```
typedef enum _CpaDcSessionState CpaDcSessionState
```

Supported session state settings

Description:

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 callback function has completed.

4.4.3.11 CpaDcCompLvl

```
typedef enum _CpaDcCompLvl CpaDcCompLvl
```

Supported compression levels

Description:

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

4.4.3.12 CpaDcReqStatus

```
typedef enum _CpaDcReqStatus CpaDcReqStatus
```

Supported additional details from accelerator

Description:

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

4.4.3.13 CpaDcAutoSelectBest

```
typedef enum _CpaDcAutoSelectBest CpaDcAutoSelectBest
```

Supported modes for automatically selecting the best compression type.

Description:

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

When CPA_DC_ASB_ENABLED is used the output will be a format compliant block, whether the data is compressed or not.

The following values are deprecated and should not be used. They will be removed in a future version of this file.

- CPA_DC_ASB_STATIC_DYNAMIC
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS

4.4.3.14 CpaDcSkipMode

```
typedef enum _CpaDcSkipMode CpaDcSkipMode
```

Supported modes for skipping regions of input or output buffers.

Description:

This enumeration lists the supported modes for skipping regions of input or output buffers.

4.4.3.15 CpaDcCallbackFn

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

Definition of callback function invoked for asynchronous cpaDc requests.

Description:

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, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Return values

None	

Precondition

Component has been initialized.

Postcondition

None

Note

None

See also

None

Definition at line 578 of file cpa_dc.h.

4.4.3.16 CpaDcInstanceCapabilities

```
typedef struct _CpaDcInstanceCapabilities CpaDcInstanceCapabilities
```

Implementation Capabilities Structure

Description:

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.

4.4.3.17 CpaDcSessionSetupData

```
typedef struct _CpaDcSessionSetupData CpaDcSessionSetupData
```

Session Setup Data.

Description:

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.

4.4.3.18 CpaDcSessionUpdateData

```
typedef struct _CpaDcSessionUpdateData CpaDcSessionUpdateData
```

Session Update Data.

Description:

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

4.4.3.19 CpaDcStats

```
typedef struct _CpaDcStats CpaDcStats
```

Compression Statistics Data.

Description:

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.

4.4.3.20 CpaDcRqResults

```
typedef struct _CpaDcRqResults CpaDcRqResults
```

Request results data

Description:

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.

4.4.3.21 CpaDcIntegrityCrcSize

```
typedef enum _CpaDcIntegrityCrcSize CpaDcIntegrityCrcSize
```

Integrity CRC Size

Description:

Enum of possible integrity CRC sizes.

4.4.3.22 CpaIntegrityCrc

```
typedef struct _CpaIntegrityCrc CpaIntegrityCrc
```

Integrity CRC calculation details

Description:

This structure contains information about resulting integrity CRC calculations performed for a single request.

4.4.3.23 CpaIntegrityCrc64b

```
typedef struct _CpaIntegrityCrc64b CpaIntegrityCrc64b
```

Integrity CRC64 calculation details

Description:

This structure contains information about resulting integrity CRC64 calculations performed for a single request.

4.4.3.24 CpaCrcData

```
typedef struct _CpaCrcData CpaCrcData
```

Collection of CRC related data

Description:

This structure contains data facilitating CRC calculations. After successful request, this structure will contain all resulting CRCs. Integrity specific CRCs (when enabled/supported) are located in 'CpaIntegrityCrc integrityCrc' field for 32bit values and in 'CpaIntegrityCrc64b integrityCrC64b' field for 64 bit values. Integrity CRCs cannot be accumulated across multiple requests and do not provide seeding capabilities.

Note

this structure must be allocated in physical contiguous memory

4.4.3.25 CpaDcSkipData

```
typedef struct _CpaDcSkipData CpaDcSkipData
```

Skip Region Data.

Description:

This structure contains data relating to configuring skip region behaviour. A skip region is a region of an input buffer that should be omitted from processing or a region that should be inserted into the output buffer.

4.4.3.26 CpaDcOpData

```
typedef struct _CpaDcOpData CpaDcOpData
```

(De)Compression request input parameters.

Description:

This structure contains the request information for use with compression operations.

4.4.3.27 CpaDcInstanceNotificationCbFunc

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

Callback function for instance notification support.

Description:

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(),

Definition at line 2917 of file cpa_dc.h.

4.4.4 Enumeration Type Documentation

4.4.4.1 _CpaDcFlush

`enum _CpaDcFlush`

Supported flush flags

Description:

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.

Definition at line 154 of file cpa_dc.h.

4.4.4.2 _CpaDcHuffType

enum _CpaDcHuffType

Supported Huffman Tree types

Description:

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.

Enumerator

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

Definition at line 190 of file cpa_dc.h.

4.4.4.3 _CpaDcCompType

enum _CpaDcCompType

Supported compression types

Description:

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_DEFLATE	Deflate Compression
CPA_DC_LZ4	LZ4 Compression
CPA_DC_LZ4S	LZ4S Compression

Definition at line 211 of file cpa_dc.h.

4.4.4.4 _CpaDcCompWindowSize

enum _CpaDcCompWindowSize

Support for defined algorithm window sizes

Description:

This enumerated list defines the valid window sizes that can be used with the supported algorithms

Enumerator

CPA_DC_WINSIZE_4K	Window size of 4KB
CPA_DC_WINSIZE_8K	Window size of 8KB
CPA_DC_WINSIZE_16K	Window size of 16KB
CPA_DC_WINSIZE_32K	Window size of 32KB

Definition at line 230 of file cpa_dc.h.

4.4.4.5 _CpaDcCompMinMatch

enum _CpaDcCompMinMatch

Min match size in bytes

Description:

This is the min match size that will be used for the search algorithm. It is only configurable for LZ4S.

Enumerator

CPA_DC_MIN_3_BYTE_MATCH	Min Match of 3 bytes
CPA_DC_MIN_4_BYTE_MATCH	Min Match of 4 bytes

Definition at line 250 of file cpa_dc.h.

4.4.4.6 _CpaDcCompLZ4BlockMaxSize

enum _CpaDcCompLZ4BlockMaxSize

Maximum LZ4 output block size

Description:

Maximum LZ4 output block size

Enumerator

CPA_DC_LZ4_MAX_BLOCK_SIZE_64K	Maximum block size 64K
CPA_DC_LZ4_MAX_BLOCK_SIZE_256K	Maximum block size 256K
CPA_DC_LZ4_MAX_BLOCK_SIZE_1M	Maximum block size 1M
CPA_DC_LZ4_MAX_BLOCK_SIZE_4M	Maximum block size 4M

Definition at line 265 of file cpa_dc.h.

4.4.4.7 _CpaDcChecksum

`enum _CpaDcChecksum`

Supported checksum algorithms

Description:

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

Enumerator

CPA_DC_NONE	No checksum required
CPA_DC_CRC32	Application requires a CRC32 checksum
CPA_DC_ADLER32	Application requires Adler-32 checksum
CPA_DC_CRC32_ADLER32	Application requires both CRC32 and Adler-32 checksums
CPA_DC_XXHASH32	Application requires xxHash-32 checksum

Definition at line 287 of file cpa_dc.h.

4.4.4.8 _CpaDcSessionDir

`enum _CpaDcSessionDir`

Supported session directions

Description:

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 for both compression and decompression

Definition at line 312 of file cpa_dc.h.

4.4.4.9 _CpaDcSessionState

`enum _CpaDcSessionState`

Supported session state settings

Description:

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 callback 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

Definition at line 341 of file cpa_dc.h.

4.4.4.10 _CpaDcCompLvl

`enum _CpaDcCompLvl`

Supported compression levels

Description:

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

Enumerator

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
CPA_DC_L10	Compression level 10
CPA_DC_L11	Compression level 11
CPA_DC_L12	Compression level 12

Definition at line 363 of file cpa_dc.h.

4.4.4.11 _CpaDcReqStatus

`enum _CpaDcReqStatus`

Supported additional details from accelerator

Description:

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. This is an indication that output buffer has overflowed. For stateful sessions, this is a warning (the input can be adjusted and resubmitted). For stateless sessions this is an error condition
CPA_DC_SOFTERR	Other non-fatal detected
CPA_DC_FATALERR	Fatal error detected
CPA_DC_MAX_RESUBITERR	On an error being detected, the firmware attempted to correct and resubmitted the request, however, the maximum resubmit value was exceeded

Enumerator

CPA_DC_INCOMPLETE_FILE_ERR	The input file is incomplete. Note this is an indication that the request was submitted with a CPA_DC_FLUSH_FINAL, however, a BFINAL bit was not found in the request
CPA_DC_WDOG_TIMER_ERR	The request was not completed as a watchdog timer hardware event occurred
CPA_DC_EP_HARDWARE_ERR	Request was not completed as an end point hardware error occurred (for example, a parity error)
CPA_DC_VERIFY_ERROR	Error detected during "compress and verify" operation
CPA_DC_EMPTY_DYM_BLK	Decompression request contained an empty dynamic stored block (not supported)
CPA_DC_CRC_INTEG_ERR	A data integrity CRC error was detected
CPA_DC_LZ4_MAX_BLOCK_SIZE_EXCEEDED	LZ4 max block size exceeded
CPA_DC_LZ4_BLOCK_OVERFLOW_ERR	LZ4 Block Overflow Error
CPA_DC_LZ4_TOKEN_IS_ZERO_ERR	LZ4 Decoded token offset or token length is zero
CPA_DC_LZ4_DISTANCE_OUT_OF_RANGE_ERR	LZ4 Distance out of range for len/distance pair

Definition at line 403 of file cpa_dc.h.

4.4.4.12 _CpaDcAutoSelectBest

```
enum _CpaDcAutoSelectBest
```

Supported modes for automatically selecting the best compression type.

Description:

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

When CPA_DC_ASB_ENABLED is used the output will be a format compliant block, whether the data is compressed or not.

The following values are deprecated and should not be used. They will be removed in a future version of this file.

- CPA_DC_ASB_STATIC_DYNAMIC
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS

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
CPA_DC_ASB_ENABLED	Auto select best mode is enabled

Definition at line 484 of file cpa_dc.h.

4.4.4.13 _CpaDcSkipMode

enum _CpaDcSkipMode

Supported modes for skipping regions of input or output buffers.

Description:

This enumeration lists the supported modes for skipping regions of input or output buffers.

Enumerator

CPA_DC_SKIP_DISABLED	Skip mode is disabled
CPA_DC_SKIP_AT_START	Skip region is at the start of the buffer.
CPA_DC_SKIP_AT_END	Skip region is at the end of the buffer.
CPA_DC_SKIP_STRIDE	Skip region occurs at regular intervals within the buffer. CpaDcSkipData.strideLength specifies the number of bytes between each skip region.

Definition at line 510 of file cpa_dc.h.

4.4.4.14 _CpaDcIntegrityCrcSize

enum _CpaDcIntegrityCrcSize

Integrity CRC Size

Description:

Enum of possible integrity CRC sizes.

Enumerator

CPA_DC_INTEGRITY_CRC32	32-bit Integrity CRCs
CPA_DC_INTEGRITY_CRC64	64-bit integrity CRCs

Definition at line 901 of file cpa_dc.h.

4.4.5 Function Documentation

4.4.5.1 cpaDcQueryCapabilities()

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

Retrieve Instance Capabilities

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

Only a synchronous version of this function is provided.

See also

None

4.4.5.2 cpaDcInitSession()

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

Initialize compression decompression session

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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 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

4.4.5.3 cpaDcResetSession()

```
CpaStatus cpaDcResetSession (
            const CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle )
```

Compression Session Reset Function.

Description:

This function will reset a previously initialized session handle Reset will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the reset function at a later time.

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_UNSUPPORTED	Function is not supported.

Precondition

The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcInitSession function.

Postcondition

None

Note

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

See also

cpaDcInitSession()

4.4.5.4 cpaDcResetXXHashState()

```
CpaStatus cpaDcResetXXHashState (
            const CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle )
```

Reset of the xxHash internal state on a session.

Description:

This function will reset the internal xxHash state maintained within a session. This would be used in conjunction with the CpaDcSessionSetupData.accumulateXXHash flag being set to TRUE for this session. It will enable reseting (reinitialising) just the xxHash calculation back to the state when the session was first initialised.

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_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcInitSession function.

Postcondition

None

Note

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

See also

4.4.5.5 cpaDcUpdateSession()

```
CpaStatus cpaDcUpdateSession (
            const CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle,
            CpaDcSessionUpdateData * pSessionUpdateData )
```

Compression Session Update Function.

Description:

This function is used to modify some select compression parameters of a previously initialized session handle. Th update will fail if resources required for the new session settings are not available. Specifically, this function may fail if no intermediate buffers are associated with the instance, and the intended change would require these buffers. This function can be called at any time after a successful call of cpaDcInitSession(). This function does not change the parameters to compression request already in flight.

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.
`in`	*pSessionUpdateData*	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

The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcInitSession function.

Postcondition

None

Note

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

See also

cpaDcInitSession()

4.4.5.6 cpaDcRemoveSession()

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

Compression Session Remove Function.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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()

4.4.5.7 cpaDcDeflateCompressBound()

```
CpaStatus cpaDcDeflateCompressBound (
            const CpaInstanceHandle dcInstance,
            CpaDcHuffType huffType,
            Cpa32U inputSize,
            Cpa32U * outputSize )
```

Deflate Compression Bound API

Description:

This function provides the maximum output buffer size for a Deflate compression operation in the "worst case" (non-compressible) scenario. It's primary purpose is for output buffer memory allocation.

Context:

This is a synchronous function that will not 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	*huffType*	CpaDcHuffType to be used with this operation.
in	*inputSize*	Input Buffer size.
out	*outputSize*	Maximum output buffer size.

Return values

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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. The cpaDcDeflateCompressBound() API is intended to reduce the likelihood of overflow occurring during compression operations. An overflow may occur in some exception cases.

See also

None

4.4.5.8 cpaDcLZ4CompressBound()

```
CpaStatus cpaDcLZ4CompressBound (
            const CpaInstanceHandle dcInstance,
            Cpa32U inputSize,
            Cpa32U * outputSize )
```

LZ4 Compression Bound API

Description:

This function provides the maximum output buffer size for a LZ4 compression operation in the "worst case" (non-compressible) scenario. It's primary purpose is for output buffer memory allocation.

Context:

This is a synchronous function that will not 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	*inputSize*	Input Buffer size.
out	*outputSize*	Maximum output buffer size.

Return values

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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

None

4.4.5.9 cpaDcLZ4SCompressBound()

```
CpaStatus cpaDcLZ4SCompressBound (
            const CpaInstanceHandle dcInstance,
            Cpa32U inputSize,
            Cpa32U * outputSize )
```

LZ4S Compression Bound API

Description:

This function provides the maximum output buffer size for a LZ4S compression operation in the "worst case" (non-compressible) scenario. It's primary purpose is for output buffer memory allocation.

Context:

This is a synchronous function that will not 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	*inputSize*	Input Buffer size.
out	*outputSize*	Maximum output buffer size.

Return values

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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

None

4.4.5.10 cpaDcCompressData()

```
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.

Description:

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

Context:

When called as an asynchronous function 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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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 stateful 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.

When an instance supports compressAndVerifyAndRecover, it is enabled by default when using cpaDcCompressData(). If this feature needs to be disabled, cpaDcCompressData2() must be used.

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

4.4.5.11 cpaDcCompressData2()

```
CpaStatus cpaDcCompressData2 (
            CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle,
            CpaBufferList * pSrcBuff,
            CpaBufferList * pDestBuff,
            CpaDcOpData * pOpData,
            CpaDcRqResults * pResults,
            void * callbackTag )
```

Submit a request to compress a buffer of data.

Description:

This API consumes data from the input buffer and generates compressed data in the output buffer. This API is very similar to cpaDcCompressData() except it provides a CpaDcOpData structure for passing additional input parameters not covered in cpaDcCompressData().

Context:

When called as an asynchronous function 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	*pOpData*	Additional parameters.
in,out	*pResults*	Pointer to results structure
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.

Return values

CPA_DC_BAD_DATA	The input data was not properly formed.
CPA_STATUS_UNSUPPORTED	Function is not supported.
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

See also

cpaDcCompressData()

4.4.5.12 cpaDcNsCompressData()

```
CpaStatus cpaDcNsCompressData (
              CpaInstanceHandle dcInstance,
              CpaDcNsSetupData * pSetupData,
              CpaBufferList * pSrcBuff,
              CpaBufferList * pDestBuff,
              CpaDcOpData * pOpData,
              CpaDcRqResults * pResults,
              CpaDcCallbackFn callbackFn,
              void * callbackTag )
```

Submit a request to compress a buffer of data without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcCompressData function.

Description:

This API consumes data from the input buffer and generates compressed data in the output buffer. Unlike the other compression APIs this does not use a previously created session. This is a "one-shot" API that requests can be directly submitted to.

Context:

When called as an asynchronous function 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	*pSetupData*	Configuration structure for compression.
in	*pSrcBuff*	Pointer to data buffer for compression.
in	*pDestBuff*	Pointer to buffer space for data after compression.
in	*pOpData*	Additional input parameters.
in,out	*pResults*	Pointer to results structure
in	*callbackFn*	For synchronous operation this callback shall be a null pointer.
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_STATUS_UNSUPPORTED	Function is not supported.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition

None

Postcondition

None

Note

This function passes control to the compression service for processing

Checksum rules. The checksum rules are the same as those for the session based APIs (cpaDcCompressData or cpaDcCompressData2) with the following exception.

1. If the algorithm specified is CPA_DC_LZ4 or CPA_DC_LZ4S the xxHash32 checksum will not be maintained across calls to the API. The implication is that the xxHash32 value will only be valid for the output of a single request, no state will be saved. If an LZ4 frame is required, even in recoverable error scenarios such as CPA_DC_OVERFLOW, the checksum will not be continued. If that is required the session based API must be used.

See also

None

4.4.5.13 cpaDcDecompressData()

```
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.

Description:

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

Context:

When called as an asynchronous function 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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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()

4.4.5.14 cpaDcDecompressData2()

```
CpaStatus cpaDcDecompressData2 (
            CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle,
            CpaBufferList * pSrcBuff,
            CpaBufferList * pDestBuff,
            CpaDcOpData * pOpData,
            CpaDcRqResults * pResults,
            void * callbackTag )
```

Submit a request to decompress a buffer of data.

Description:

This API consumes compressed data from the input buffer and generates uncompressed data in the output buffer. This API is very similar to cpaDcDecompressData() except it provides a CpaDcOpData structure for passing additional input parameters not covered in cpaDcDecompressData().

Context:

When called as an asynchronous function 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	*pOpData*	Additional input parameters.
in,out	*pResults*	Pointer to results structure
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_UNSUPPORTED	Function is not supported.
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.

See also

cpaDcDecompressData() cpaDcCompressData2() cpaDcCompressData()

4.4.5.15 cpaDcNsDecompressData()

```
CpaStatus cpaDcNsDecompressData (
            CpaInstanceHandle dcInstance,
            CpaDcNsSetupData * pSetupData,
            CpaBufferList * pSrcBuff,
            CpaBufferList * pDestBuff,
            CpaDcOpData * pOpData,
            CpaDcRqResults * pResults,
            CpaDcCallbackFn callbackFn,
            void * callbackTag )
```

Submit a request to decompress a buffer of data without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcDecompressData function.

Description:

This API consumes data from the input buffer and generates decompressed data in the output buffer. Unlike the other decompression APIs this does not use a previously created session. This is a "one-shot" API that requests can be directly submitted to.

Context:

When called as an asynchronous function 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	*pSetupData*	Configuration structure for decompression..
in	*pSrcBuff*	Pointer to data buffer for decompression.
in	*pDestBuff*	Pointer to buffer space for data after decompression.
in	*pOpData*	Additional input parameters.
in,out	*pResults*	Pointer to results structure
in	*callbackFn*	For synchronous operation this callback shall be a null pointer.
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_STATUS_UNSUPPORTED	Function is not supported.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition

None

Postcondition

None

Note

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

See also

cpaDcDecompressData() cpaDcCompressData2() cpaDcCompressData()

4.4.5.16 cpaDcGenerateHeader()

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

Generate compression header.

Description:

This function generates the gzip header, zlib header or LZ4 frame header and stores it in the destination buffer. The type of header created is determined using the compression algorithm selected using CpaDcSessionSetupData.compType, for the session associated with the session handle.

Context:

This function may be call from any context.

Assumptions:

None

Side-Effects:

None

Blocking:

No

Reentrant:

No

Thread-safe:

Yes

Parameters

in	*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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

pSessionHandle has been setup using cpaDcInitSession()

Note

When the deflate compression algorithm is used, 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 Gzip or a Zlib header the session must have been initialized with CpaDcCompType CPA_DC_DEFLATE. 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)

When LZ4 algorithm is used, this function can output a 7 byte frame header. This function will set the LZ4 frame header with:

- Magic number 0x184D2204
- The LZ4 max block size defined in the CpaDcSessionSetupData
- Flag byte as:
  - Version = 1
  - Block independence = 0
  - Block checksum = 0

  - Content size present = 0
  - Content checksum present = 1
  - Dictionary ID present = 0
- Content size = 0
- Dictionary ID = 0
- Header checksum = 1 byte representing the second byte of the XXH32 of the frame decriptor field.

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

For any of the compression algorithms used, the application is responsible to offset the pData pointer in CpaBufferList by the length of the header before calling the CpaDcCompressData() or CpaDcCompressData2() functions.

See also

None

4.4.5.17 cpaDcGenerateFooter()

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

Generate compression footer.

Description:

This function generates the footer for gzip, zlib or LZ4. The generated footer is stored it in the destination buffer. The type of footer created is determined using the compression algorithm selected for the session associated with the session handle.

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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.

For LZ4 compression, this function adds the LZ4 frame footer using XXH32 algorithm of the uncompressed data. The XXH32 checksum is added after the end mark. This section is defined in the documentation of the LZ4 frame format at: https://github.com/lz4/lz4/blob/dev/doc/lz4_Frame_format.md

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. It is necessary for the application to ensure that there is always sufficient memory in the destination buffer to append the footer. In the event that the destination buffer would be too small to accept the footer, overflow will not be reported.

See also

None

4.4.5.18 cpaDcNsGenerateHeader()

```
CpaStatus cpaDcNsGenerateHeader (
            CpaDcNsSetupData * pSetupData,
            CpaFlatBuffer * pDestBuff,
            Cpa32U * count )
```

Generate compression header without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcGenerateHeader function.

Description:

This API generates the required compression format header and stores it in the output buffer.

Context:

This function may be called from any context.

Assumptions:

None

Side-Effects:

None

Blocking:

No

Reentrant:

No

Thread-safe:

Yes

Parameters

in	*pSetupData*	Pointer to Ns Configuration structure.
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.

Return values

CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Note

This function outputs the required compression format header to the destination buffer. The CpaDcNsSetupData structure fields are used to determine the header type.

To output an LZ4 header the structure must have been initialized with with CpaDcCompType CPA_DC_LZ4. To output a gzip or zlib header the structure must have been initialized with CpaDcCompType CPA_DC_DEFLATE. To output a gzip header the structure must have been initialized with CpaDcChecksum CPA_DC_CRC32. To output a zlib header the structure must have been initialized with CpaDcChecksum CPA_DC_ADLER32. For CpaDcChecksum CPA_DC_NONE no header is output.

The counter parameter will be set to the number of bytes added to the buffer.

See also

cpaDcGenerateHeader

4.4.5.19 cpaDcNsGenerateFooter()

```
CpaStatus cpaDcNsGenerateFooter (
            CpaDcNsSetupData * pSetupData,
            Cpa64U totalLength,
            CpaFlatBuffer * pDestBuff,
            CpaDcRqResults * pResults )
```

Generate compression footer without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcGenerateFooter function.

Description:

This API generates the footer for the required format and stores it in the destination 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	*pSetupData*	Pointer to Ns Configuration structure.
in	*totalLength*	Total accumulated length of input data processed. See description for formats that make use of this parameter.
in	*pDestBuff*	Pointer to data buffer where the compression footer will go.
in,out	*pResults*	Pointer to results structure filled by CpaDcNsCompressData. 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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

pResults structure has been filled by CpaDcNsCompressData().

Note

This function outputs the required compression format footer to the destination buffer. The CpaDcNsSetupData structure fields are used to determine the footer type created.

To output an LZ4 footer the structure must have been initialized with with CpaDcCompType CPA_DC_LZ4. To output a gzip or zlib footer the structure must have been initialized with CpaDcCompType CPA_DC_DEFLATE. To output a gzip footer the structure must have been initialized with CpaDcChecksum CPA_DC_CRC32 and the totalLength parameter initialized to the total accumulated length of data processed. To output a zlib footer the structure must have been initialized with CpaDcChecksum CPA_DC_ADLER32. For CpaDcChecksum CPA_DC_NONE no footer is output.

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

See also

CpaDcNsSetupData cpaDcNsGenerateHeader cpaDcGenerateFooter

4.4.5.20 cpaDcGetStats()

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

Retrieve statistics

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

See also

None

4.4.5.21 cpaDcGetNumInstances()

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

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

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

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

See also

cpaDcGetInstances

4.4.5.22 cpaDcGetInstances()

```
CpaStatus cpaDcGetInstances (
            Cpa16U numInstances,
            CpaInstanceHandle * dcInstances )
```

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

Description:

```
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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

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

See also

cpaDcGetInstances

4.4.5.23 cpaDcGetNumIntermediateBuffers()

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

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

Description:

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 that will use dynamic compression.

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

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

See also

cpaDcStartInstance()

4.4.5.24 cpaDcStartInstance()

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

Compression Component Initialization and Start function.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

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

See also

cpaDcStopInstance() cpaDcGetNumIntermediateBuffers()

4.4.5.25 cpaDcStopInstance()

```
CpaStatus cpaDcStopInstance (
            CpaInstanceHandle instanceHandle )
```

Compress Component Stop function.

Description:

This function will stop the Compression component and free all system resources associated with it. The client MUST ensure that all outstanding operations have completed before calling this function. The recommended approach to ensure this is to deregister all session or callback handles before calling this function. If outstanding operations still exist when this function is invoked, the callback function for each of those operations will NOT be invoked and the shutdown will continue. If the component is to be restarted, then a call to cpaDcStartInstance is required.

Context:

This function may sleep, and so 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	*instanceHandle*	Handle to an instance of this API to be shutdown.

Return values

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Suggested course of action is to ensure requests are not still being submitted and that all sessions are deregistered. If this does not help, then forcefully remove the component from the system.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

The component has been initialized via cpaDcStartInstance

Postcondition

None

Note

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

See also

cpaDcStartInstance()

4.4.5.26 cpaDcInstanceGetInfo2()

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

Function to get information on a particular instance.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

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

Postcondition

None

Note

None

See also

cpaDcGetNumInstances, cpaDcGetInstances, CpaInstanceInfo2

4.4.5.27 cpaDcInstanceSetNotificationCb()

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

Subscribe for instance notifications.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

Instance has been initialized.

Postcondition

None

Note

None

See also

CpaDcInstanceNotificationCbFunc

4.4.5.28 cpaDcGetSessionSize()

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

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

Description:

```
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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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 and may be zero.

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()

4.4.5.29 cpaDcBufferListGetMetaSize()

```
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.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

None

See also

cpaDcGetInstances()

4.4.5.30 cpaDcGetStatusText()

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

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

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

None

See also

CpaStatus

4.4.5.31 cpaDcSetAddressTranslation()

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

Set Address Translation function

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

See also

None

4.4.5.32 cpaDcDpGetSessionSize()

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

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

Description:

```
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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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()

4.4.5.33 cpaDcDpUpdateSession()

```
CpaStatus cpaDcDpUpdateSession (
            const CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle,
            CpaDcSessionUpdateData * pSessionUpdateData )
```

Compression Session Update Function.

Description:

This function is used to modify some select compression parameters of a previously initialized session handlei for a data plane session. Th update will fail if resources required for the new session settings are not available. Specifically, this function may fail if no intermediate buffers are associated with the instance, and the intended change would require these buffers. This function can be called at any time after a successful call of cpaDcDpInitSession(). This function does not change the parameters to compression request already in flight.

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:

No

Parameters

`in`	*dcInstance*	Instance handle.
`in,out`	*pSessionHandle*	Session handle.
`in`	*pSessionUpdateData*	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

The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcDpInitSession function.

Postcondition

None

Note

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

See also

cpaDcDpInitSession()

4.4.5.34 cpaDcDpRemoveSession()

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

Compression Data Plane Session Remove Function.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

The component has been initialized via cpaDcStartInstance function.

Postcondition

None

Note

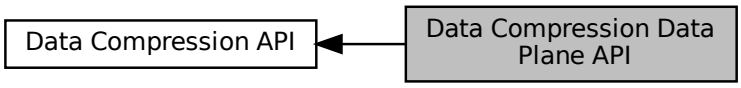
This is a synchronous function and has no completion callback associated with it.

See also

cpaDcDpInitSession

4.5 Data Compression Data Plane API

Collaboration diagram for Data Compression Data Plane API:



Data Structures

- struct _CpaDcDpOpData

Typedefs

- typedef struct _CpaDcDpOpData CpaDcDpOpData
- typedef void(∗ CpaDcDpCallbackFn) (CpaDcDpOpData ∗pOpData)

Functions

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

4.5.1 Detailed Description

File: cpa_dc_dp.h

Description:

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).

4.5.2 Typedef Documentation

4.5.2.1 CpaDcDpOpData

```
typedef struct _CpaDcDpOpData CpaDcDpOpData
```

Operation Data for compression data plane API.

Description:

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

4.5.2.2 CpaDcDpCallbackFn

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

Definition of callback function for compression data plane API.

Description:

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 asynchronous 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

Definition at line 244 of file cpa_dc_dp.h.

4.5.3 Function Documentation

4.5.3.1 cpaDcDpInitSession()

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

Initialize compression or decompression data plane session.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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

4.5.3.2 cpaDcDpRegCbFunc()

```
CpaStatus cpaDcDpRegCbFunc (
            const CpaInstanceHandle dcInstance,
            const CpaDcDpCallbackFn pNewCb )
```

Registration of the operation completion callback function.

Description:

This function allows a completion callback function to be registered. The registered callback function is invoked on completion of asynchronous 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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

Instance has been initialized.

Postcondition

None

Note

None

See also

cpaDcDpCbFunc

4.5.3.3 cpaDcDpEnqueueOp()

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

Enqueue a single compression or decompression request.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

The session identified by pOpData->pSessionHandle was setup using cpaDcDpInitSession OR pOpData->pSetupData data structure was initialized for No-Session (Ns) usage. 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

4.5.3.4 cpaDcDpEnqueueOpBatch()

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

Enqueue multiple requests to the compression data plane API.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

The session identified by pOpData[i]->pSessionHandle was setup using cpaDcDpInitSession OR pOpData[i]->pSetupData data structure was initialized for No-Session (Ns) usage. 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

4.5.3.5 cpaDcDpPerformOpNow()

```
CpaStatus cpaDcDpPerformOpNow (
            CpaInstanceHandle dcInstance )
```

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

Description:

This function triggers processing of previously enqueued 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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

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

Postcondition

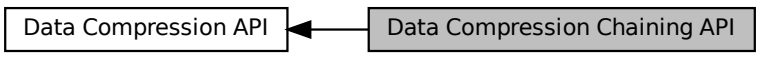
None

See also

cpaDcDpEnqueueOp, cpaDcDpEnqueueOpBatch

4.6 Data Compression Chaining API

Collaboration diagram for Data Compression Chaining API:



Data Structures

- struct _CpaDcChainSessionSetupData
- struct _CpaDcChainOpData
- struct _CpaDcChainRqResults

Typedefs

- typedef enum _CpaDcChainOperations CpaDcChainOperations
- typedef enum _CpaDcChainSessionType CpaDcChainSessionType
- typedef struct _CpaDcChainSessionSetupData CpaDcChainSessionSetupData
- typedef struct _CpaDcChainOpData CpaDcChainOpData
- typedef struct _CpaDcChainRqResults CpaDcChainRqResults

Enumerations

- enum _CpaDcChainOperations
- enum _CpaDcChainSessionType

Functions

- CpaStatus cpaDcChainGetSessionSize (CpaInstanceHandle dcInstance, CpaDcChainOperations operation, Cpa8U numSessions, CpaDcChainSessionSetupData *pSessionData, Cpa32U *pSessionSize)
- CpaStatus cpaDcChainInitSession (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaDcChainOperations operation, Cpa8U numSessions, CpaDcChainSessionSetupData *pSessionData, CpaDcCallbackFn callbackFn)
- CpaStatus cpaDcChainResetSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)
- CpaStatus cpaDcChainRemoveSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)
- CpaStatus cpaDcChainPerformOp (CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcChainOperations operation, Cpa8U numOpDatas, CpaDcChainOpData *pChainOpData, CpaDcChainRqResults *pResults, void *callbackTag)

4.6.1 Detailed Description

File: cpa_dc_chain.h

Description:

These functions specify the API for Data Compression Chaining operations.

Remarks

4.6.2 Typedef Documentation

4.6.2.1 CpaDcChainOperations

```
typedef enum _CpaDcChainOperations CpaDcChainOperations
```

Supported operations for compression chaining

Description:

This enumeration lists the supported operations for compression chaining

4.6.2.2 CpaDcChainSessionType

```
typedef enum _CpaDcChainSessionType CpaDcChainSessionType
```

Supported session types for data compression chaining.

Description:

This enumeration lists the supported session types for data compression chaining.

4.6.2.3 CpaDcChainSessionSetupData

```
typedef struct _CpaDcChainSessionSetupData CpaDcChainSessionSetupData
```

Chaining Session Setup Data.

Description:

This structure contains data relating to set up chaining sessions. The client needs to complete the information in this structure in order to setup chaining sessions.

4.6.2.4 CpaDcChainOpData

```
typedef struct _CpaDcChainOpData CpaDcChainOpData
```

Compression chaining request input parameters.

Description:

This structure contains the request information to use with compression chaining operations.

4.6.2.5 CpaDcChainRqResults

```
typedef struct _CpaDcChainRqResults CpaDcChainRqResults
```

Chaining request results data

Description:

This structure contains the request results.

4.6.3 Enumeration Type Documentation

4.6.3.1 _CpaDcChainOperations

```
enum _CpaDcChainOperations
```

Supported operations for compression chaining

Description:

This enumeration lists the supported operations for compression chaining

Enumerator

CPA_DC_CHAIN_COMPRESS_THEN_HASH	2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform hash on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for hash setup data
CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT	2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform encryption on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for encryption setup data
CPA_DC_CHAIN_COMPRESS_THEN_HASH_EN↩ CRYPT	2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform hash on compressed text and encryption on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for hash and encryption setup data
CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT↩ _HASH	2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform encryption on compressed text and hash on compressed & encrypted text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for encryption and hash setup data
CPA_DC_CHAIN_COMPRESS_THEN_AEAD	2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform AEAD encryption on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for AEAD encryption setup data
CPA_DC_CHAIN_HASH_THEN_COMPRESS	2 operations for chaining: 1st operation is to perform hash on plain text 2nd operation is to perform compression on plain text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for hash setup data 2nd entry is for compression setup data
CPA_DC_CHAIN_HASH_VERIFY_THEN_DECO↩ MPRESS	2 operations for chaining: 1st operation is to perform hash verify on compressed text 2nd operation is to perform decompression on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for hash setup data 2nd entry is for decompression setup data
CPA_DC_CHAIN_DECRYPT_THEN_DECOMPR↩ ESS	2 operations for chaining: 1st operation is to perform decryption on compressed & encrypted text 2nd operation is to perform decompression on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for decryption setup data 2nd entry is for decompression setup data

Enumerator

CPA_DC_CHAIN_HASH_VERIFY_DECRYPT_THEN_DECOMPRESS	2 operations for chaining: 1st operation is to perform hash verify on compressed & encrypted text and decryption on compressed & encrypted text 2nd operation is to perform decompression on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for hash and decryption setup data 2nd entry is for decompression setup data
CPA_DC_CHAIN_DECRYPT_HASH_VERIFY_THEN_DECOMPRESS	2 operations for chaining: 1st operation is to perform decryption on compressed & encrypted text and hash verify on compressed text 2nd operation is to perform decompression on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for decryption and hash setup data 2nd entry is for decompression setup data
CPA_DC_CHAIN_AEAD_THEN_DECOMPRESS	2 operations for chaining: 1st operation is to perform AEAD decryption on compressed & encrypted text 2nd operation is to perform decompression on compressed text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for AEAD decryption setup data 2nd entry is for decompression setup data
CPA_DC_CHAIN_DECOMPRESS_THEN_HASH_VERIFY	2 operations for chaining: 1st operation is to perform decompression on compressed text 2nd operation is to perform hash verify on plain text $<$ 2 entries in CpaDcChainSessionSetupData array: 1st entry is for decompression setup data 2nd entry is for hash setup data
CPA_DC_CHAIN_COMPRESS_THEN_AEAD_THEN_HASH	3 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform AEAD encryption compressed text 3rd operation is to perfom hash on compressed & encrypted text $<$ 3 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for AEAD encryption setup data 3rd entry is for hash setup data

Definition at line 48 of file cpa_dc_chain.h.

4.6.3.2 _CpaDcChainSessionType

`enum _CpaDcChainSessionType`

Supported session types for data compression chaining.

Description:

This enumeration lists the supported session types for data compression chaining.

Enumerator

CPA_DC_CHAIN_COMPRESS_DECOMPRESS	Indicate the session is for compression or decompression
CPA_DC_CHAIN_SYMMETRIC_CRYPTO	Indicate the session is for symmetric crypto

Definition at line 158 of file cpa_dc_chain.h.

4.6.4 Function Documentation

4.6.4.1 cpaDcChainGetSessionSize()

```
CpaStatus cpaDcChainGetSessionSize (
            CpaInstanceHandle dcInstance,
            CpaDcChainOperations operation,
            Cpa8U numSessions,
            CpaDcChainSessionSetupData * pSessionData,
            Cpa32U * pSessionSize )
```

Get the size of the memory required to hold the chaining sessions information.

Description:

The client of the Data Compression API is responsible for allocating sufficient memory to hold chaining sessions information. This function provides a way for determining the size of chaining sessions.

Context:

No restrictions

Assumptions:

None

Side-Effects:

None

Blocking:

No

Reentrant:

No

Thread-safe:

Yes

Parameters

in	*dcInstance*	Instance handle.
in	*operation*	The operation for chaining
in	*numSessions*	Number of sessions for the chaining
in	*pSessionData*	Pointer to an array of CpaDcChainSessionSetupData structures. There should be numSessions entries in the array.
out	*pSessionSize*	On return, this parameter will be the size of the memory that will be required by cpaDcChainInitSession() for 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_UNSUPPORTED	Function is not supported.

Precondition

None

Postcondition

None

Note

Only a synchronous version of this function is provided.

See also

cpaDcChainInitSession()

4.6.4.2 cpaDcChainInitSession()

```
CpaStatus cpaDcChainInitSession (
            CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle,
            CpaDcChainOperations operation,
            Cpa8U numSessions,
            CpaDcChainSessionSetupData * pSessionData,
            CpaDcCallbackFn callbackFn )
```

Initialize data compression chaining session

Description:

This function is used to initialize compression/decompression chaining sessions. 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`	*operation*	The operations for chaining
`in`	*numSessions*	Number of sessions for chaining
`in,out`	*pSessionData*	Pointer to an array of CpaDcChainSessionSetupData structures. There should be numSessions entries in the array.
`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.

Return values

CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

dcInstance has been started using cpaDcStartInstance.

Postcondition

None

Note

Only a synchronous version of this function is provided.

pSessionData Setup Rules

1. Each element in CpaDcChainSessionSetupData structure array provides (de)compression or a symmetric crypto session setup data.
2. The supported chaining operations are listed in CpaDcChainOperations. This enum indicates the number of operations in a chain and the order in which they are performed.
3. The order of entries in pSessionData[] should be consistent with the CpaDcChainOperations perform order. As an example, for CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT, pSessionData[0] holds the compression setup data and pSessionData[1] holds the encryption setup data..
4. The numSessions for each chaining operation are provided in the comments of enum CpaDcChainOperations.
5. For a (de)compression session, the corresponding pSessionData[]->sessType should be set to CPA_DC_CHAIN_COMPRESS_DECOMPRESS and pSessionData[]->pDcSetupData should point to a CpaDcSessionSetupData structure.
6. For a symmetric crypto session, the corresponding pSessionData[]->sessType should be set to CPA_DC_CHAIN_SYMMETRIC_CRYPTO and pSessionData[]->pCySetupData should point to a CpaCySymSessionSetupData structure.
7. Combined compression sessions are not supported for chaining.
8. Stateful compression is not supported for chaining.
9. Both CRC32 and Adler32 over the input data are supported for chaining.

See also

None

4.6.4.3 cpaDcChainResetSession()

```
CpaStatus cpaDcChainResetSession (
            const CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle )
```

Reset a compression chaining session.

Description:

This function will reset a previously initialized session handle. Reset will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the reset function at a later time.

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_UNSUPPORTED	Function is not supported.

Precondition

The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcChainInitSession function.

Postcondition

None

Note

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

See also

cpaDcChainInitSession()

4.6.4.4 cpaDcChainRemoveSession()

```
CpaStatus cpaDcChainRemoveSession (
            const CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle )
```

Remove a compression chaining session.

Description:

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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

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

cpaDcChainInitSession()

4.6.4.5 cpaDcChainPerformOp()

```
CpaStatus cpaDcChainPerformOp (
            CpaInstanceHandle dcInstance,
            CpaDcSessionHandle pSessionHandle,
            CpaBufferList * pSrcBuff,
            CpaBufferList * pDestBuff,
            CpaDcChainOperations operation,
            Cpa8U numOpDatas,
            CpaDcChainOpData * pChainOpData,
            CpaDcChainRqResults * pResults,
            void * callbackTag )
```

Submit a request to perform chaining operations.

Description:

This function is used to perform chaining operations over data from the source buffer.

Context:

When called as an asynchronous function 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 input data buffer.
out	*pDestBuff*	Pointer to output data buffer.
in	*operation*	Operation for the chaining request
in	*numOpDatas*	The entries size CpaDcChainOpData array
in	*pChainOpData*	Pointer to an array of CpaDcChainOpData structures. There should be numOpDatas entries in the array.
in,out	*pResults*	Pointer to CpaDcChainRqResults structure.
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.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition

pSessionHandle has been setup using cpaDcChainInitSession()

Postcondition

pSessionHandle has session related state information

Note

This function passes control to the compression service for chaining processing, the supported chaining operations are described in CpaDcChainOperations.

pChainOpData Setup Rules

1. Each element in CpaDcChainOpData structure array holds either a (de)compression or a symmetric crypto operation data.
2. The order of entries in pChainOpData[] must be consistent with the order of operations described for the chaining operation in CpaDcChainOperations. As an example, for CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT, pChainOpData[0] must contain the compression operation data and pChainOpData[1] must contain the encryption operation data.
3. The numOpDatas for each chaining operation are specified in the comments for the operation in CpaDcChainOperations.
4. For a (de)compression operation, the corresponding pChainOpData[]->opType should be set to CPA_DC_CHAIN_COMPRESS_DECOMPRESS and pChainOpData[]->pDcOp should point to a CpaDcOpData structure.
5. For a symmetric crypto operation, the corresponding pChainOpData[]->opType should be set to CPA_DC_CHAIN_SYMMETRIC_CRYPTO and pChainOpData[]->pCySymOp should point to a CpaCySymOpData structure.

   (a) Stateful compression is not supported for chaining.

   (b) Partial packet processing is not supported.

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

   This function has identical checksum processing rules as cpaDcCompressData(), except:

   (a) pResults->crc32 is available to application if CpaDcSessionSetupData->checksum is set to CPA_DC_CRC32

   (b) pResults->adler32 is available to application if CpaDcSessionSetupData->checksum is set to CPA_DC_ADLER32

   (c) Both pResults->crc32 and pResults->adler32 are available if CpaDcSessionSetupData->checksum is set to CPA_DC_CRC32_ADLER32

Synchronous or asynchronous operation of the API is determined by the value of the callbackFn parameter passed to cpaDcChainInitSession() 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.

This function has identical response ordering rules as cpaDcCompressData().

See also

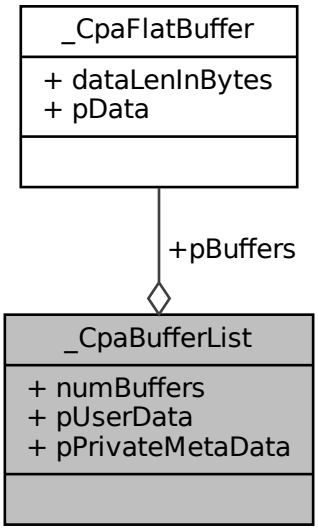
cpaDcCompressData

Chapter 5

Data Structure Documentation

5.1 _CpaBufferList Struct Reference

Collaboration diagram for _CpaBufferList:



Data Fields

- Cpa32U numBuffers
- CpaFlatBuffer * pBuffers
- void * pUserData
- void * pPrivateMetaData

5.1.1 Detailed Description

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

Description:

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.

Definition at line 164 of file cpa.h.

5.1.2 Field Documentation

5.1.2.1 numBuffers

```
Cpa32U _CpaBufferList::numBuffers
```

Number of buffers in the list

Definition at line 165 of file cpa.h.

5.1.2.2 pBuffers

```
CpaFlatBuffer* _CpaBufferList::pBuffers
```

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

Definition at line 167 of file cpa.h.

5.1.2.3 pUserData

```
void* _CpaBufferList::pUserData
```

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

Definition at line 171 of file cpa.h.

5.1.2.4 pPrivateMetaData

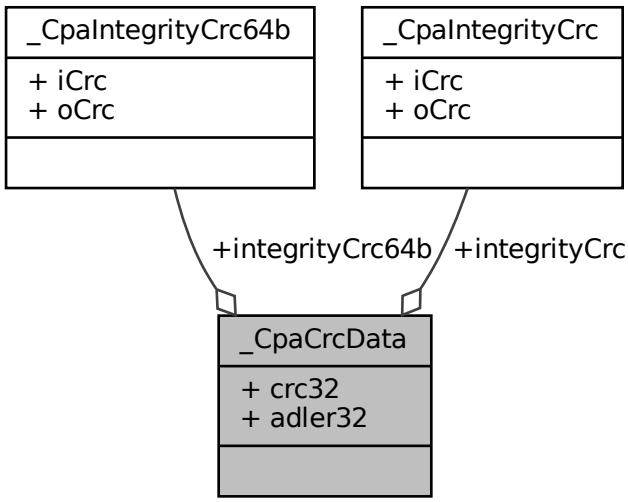
```
void* _CpaBufferList::pPrivateMetaData
```

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.

Definition at line 173 of file cpa.h.

5.2 _CpaCrcData Struct Reference

Collaboration diagram for _CpaCrcData:



Data Fields

- Cpa32U crc32
- Cpa32U adler32
- CpaIntegrityCrc integrityCrc
- CpaIntegrityCrc64b integrityCrc64b

5.2.1 Detailed Description

Collection of CRC related data

Description:

This structure contains data facilitating CRC calculations. After successful request, this structure will contain all resulting CRCs. Integrity specific CRCs (when enabled/supported) are located in 'CpaIntegrityCrc integrityCrc' field for 32bit values and in 'CpaIntegrityCrc64b integrityCrC64b' field for 64 bit values. Integrity CRCs cannot be accumulated across multiple requests and do not provide seeding capabilities.

Note

this structure must be allocated in physical contiguous memory

Definition at line 954 of file cpa_dc.h.

5.2.2 Field Documentation

5.2.2.1 crc32

`Cpa32U _CpaCrcData::crc32`

CRC32 calculated on the input buffer during compression requests and on the output buffer during decompression requests.

Definition at line 955 of file cpa_dc.h.

5.2.2.2 adler32

`Cpa32U _CpaCrcData::adler32`

ADLER32 calculated on the input buffer during compression requests and on the output buffer during decompression requests.

Definition at line 958 of file cpa_dc.h.

5.2.2.3 integrityCrc

`CpaIntegrityCrc _CpaCrcData::integrityCrc`

32bit Integrity CRCs

Definition at line 961 of file cpa_dc.h.

5.2.2.4 integrityCrc64b

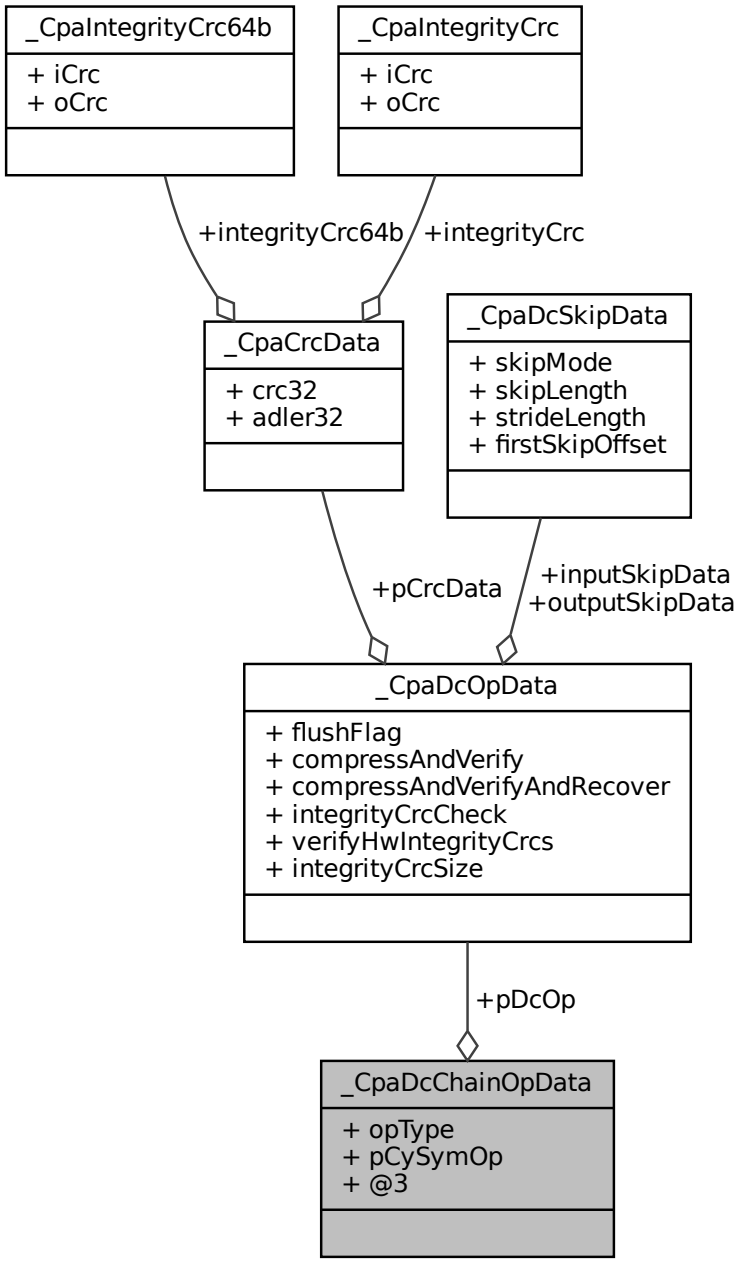
`CpaIntegrityCrc64b _CpaCrcData::integrityCrc64b`

64bit Integrity CRCs

Definition at line 963 of file cpa_dc.h.

5.3 _CpaDcChainOpData Struct Reference

Collaboration diagram for _CpaDcChainOpData:



Data Fields

- CpaDcChainSessionType opType
- union {
  CpaDcOpData * pDcOp
  CpaCySymOpData * pCySymOp
  };

5.3.1 Detailed Description

Compression chaining request input parameters.

Description:

This structure contains the request information to use with compression chaining operations.

Definition at line 196 of file cpa_dc_chain.h.

5.3.2 Field Documentation

5.3.2.1 opType

```
CpaDcChainSessionType _CpaDcChainOpData::opType
```

Indicate the type for this operation

Definition at line 197 of file cpa_dc_chain.h.

5.3.2.2 pDcOp

```
CpaDcOpData* _CpaDcChainOpData::pDcOp
```

Pointer to compression operation data

Definition at line 200 of file cpa_dc_chain.h.

5.3.2.3 pCySymOp

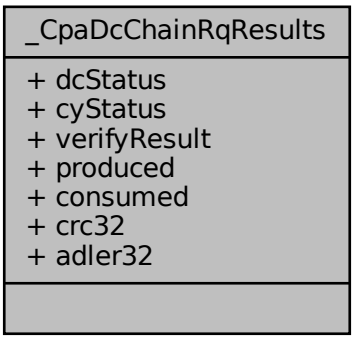
```
CpaCySymOpData* _CpaDcChainOpData::pCySymOp
```

Pointer to symmectric crypto operation data

Definition at line 202 of file cpa_dc_chain.h.

5.4 _CpaDcChainRqResults Struct Reference

Collaboration diagram for _CpaDcChainRqResults:



Data Fields

- CpaDcReqStatus dcStatus
- CpaStatus cyStatus
- CpaBoolean verifyResult
- Cpa32U produced
- Cpa32U consumed
- Cpa32U crc32
- Cpa32U adler32

5.4.1 Detailed Description

Chaining request results data

Description:

This structure contains the request results.

Definition at line 215 of file cpa_dc_chain.h.

5.4.2 Field Documentation

5.4.2.1 dcStatus

CpaDcReqStatus _CpaDcChainRqResults::dcStatus

Additional status details from compression accelerator

Definition at line 216 of file cpa_dc_chain.h.

5.4.2.2 cyStatus

CpaStatus _CpaDcChainRqResults::cyStatus

Additional status details from symmetric crypto accelerator

Definition at line 218 of file cpa_dc_chain.h.

5.4.2.3 verifyResult

CpaBoolean _CpaDcChainRqResults::verifyResult

This parameter is valid when the verifyDigest option is set in the CpaCySymSessionSetupData structure. A value of CPA_TRUE indicates that the compare succeeded. A value of CPA_FALSE indicates that the compare failed

Definition at line 220 of file cpa_dc_chain.h.

5.4.2.4 produced

Cpa32U _CpaDcChainRqResults::produced

Octets produced to the output buffer

Definition at line 225 of file cpa_dc_chain.h.

5.4.2.5 consumed

Cpa32U _CpaDcChainRqResults::consumed

Octets consumed from the input buffer

Definition at line 227 of file cpa_dc_chain.h.

5.4.2.6 crc32

```
Cpa32U _CpaDcChainRqResults::crc32
```

crc32 checksum produced by chaining operations

Definition at line 229 of file cpa_dc_chain.h.

5.4.2.7 adler32

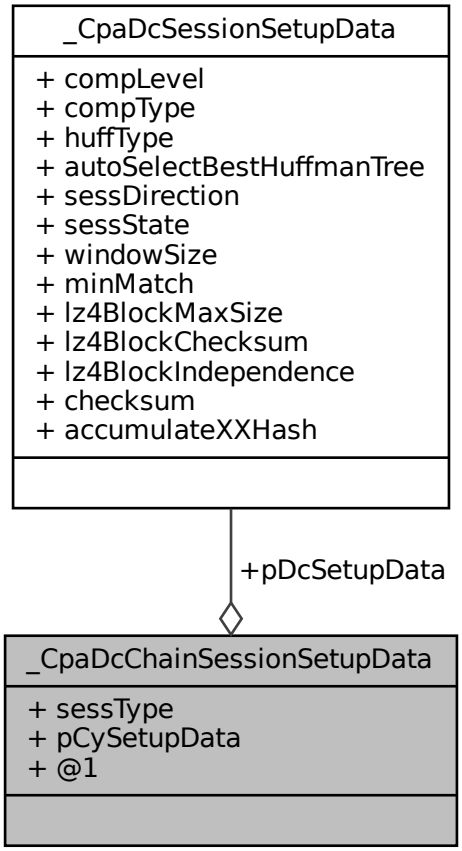
```
Cpa32U _CpaDcChainRqResults::adler32
```

adler32 checksum produced by chaining operations

Definition at line 231 of file cpa_dc_chain.h.

5.5 _CpaDcChainSessionSetupData Struct Reference

Collaboration diagram for _CpaDcChainSessionSetupData:



Data Fields

- CpaDcChainSessionType **sessType**
- union {
  CpaDcSessionSetupData * pDcSetupData
  CpaCySymSessionSetupData * pCySetupData
  };

5.5.1 Detailed Description

Chaining Session Setup Data.

Description:

This structure contains data relating to set up chaining sessions. The client needs to complete the information in this structure in order to setup chaining sessions.

Definition at line 176 of file cpa_dc_chain.h.

5.5.2 Field Documentation

5.5.2.1 pDcSetupData

```
CpaDcSessionSetupData* _CpaDcChainSessionSetupData::pDcSetupData
```

Pointer to compression session setup data

Definition at line 180 of file cpa_dc_chain.h.

5.5.2.2 pCySetupData

```
CpaCySymSessionSetupData* _CpaDcChainSessionSetupData::pCySetupData
```

Pointer to symmectric crypto session setup data

Definition at line 182 of file cpa_dc_chain.h.

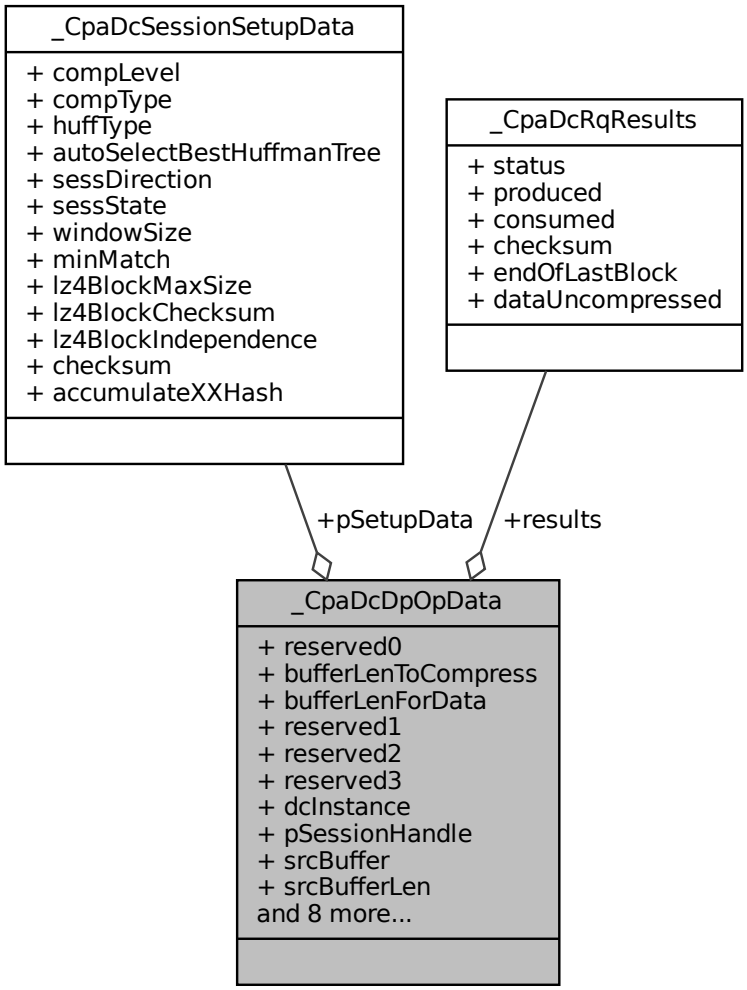
5.5.2.3 "@1

```
union { ... }
```

Indicate the type for this session

5.6 _CpaDcDpOpData Struct Reference

Collaboration diagram for _CpaDcDpOpData:



Data Fields

- Cpa64U reserved0
- Cpa32U bufferLenToCompress
- Cpa32U bufferLenForData
- Cpa64U reserved1
- Cpa64U reserved2
- Cpa64U reserved3
- CpaDcRqResults results
- CpaInstanceHandle dcInstance
- CpaDcSessionHandle pSessionHandle
- CpaPhysicalAddr srcBuffer

- Cpa32U srcBufferLen
- CpaPhysicalAddr destBuffer
- Cpa32U destBufferLen
- CpaDcSessionDir sessDirection
- CpaBoolean compressAndVerify
- CpaBoolean compressAndVerifyAndRecover
- CpaStatus responseStatus
- CpaPhysicalAddr thisPhys
- void * pCallbackTag
- CpaDcNsSetupData * pSetupData

5.6.1 Detailed Description

Operation Data for compression data plane API.

Description:

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

Definition at line 82 of file cpa_dc_dp.h.

5.6.2 Field Documentation

5.6.2.1 reserved0

```
Cpa64U _CpaDcDpOpData::reserved0
```

Reserved for internal use. Source code should not read or write this field.

Definition at line 84 of file cpa_dc_dp.h.

5.6.2.2 bufferLenToCompress

Cpa32U _CpaDcDpOpData::bufferLenToCompress

The number of bytes from the source buffer to compress. This must be less than, or more typically equal to, the total size of the source buffer (or buffer list).

Definition at line 88 of file cpa_dc_dp.h.

5.6.2.3 bufferLenForData

Cpa32U _CpaDcDpOpData::bufferLenForData

The maximum number of bytes that should be written to the destination buffer. This must be less than, or more typically equal to, the total size of the destination buffer (or buffer list).

Definition at line 94 of file cpa_dc_dp.h.

5.6.2.4 reserved1

Cpa64U _CpaDcDpOpData::reserved1

Reserved for internal use. Source code should not read or write

Definition at line 100 of file cpa_dc_dp.h.

5.6.2.5 reserved2

Cpa64U _CpaDcDpOpData::reserved2

Reserved for internal use. Source code should not read or write

Definition at line 103 of file cpa_dc_dp.h.

5.6.2.6 reserved3

Cpa64U _CpaDcDpOpData::reserved3

Reserved for internal use. Source code should not read or write

Definition at line 106 of file cpa_dc_dp.h.

5.6.2.7 results

CpaDcRqResults _CpaDcDpOpData::results

Results of the operation. Contents are valid upon completion.

Definition at line 109 of file cpa_dc_dp.h.

5.6.2.8 dcInstance

CpaInstanceHandle _CpaDcDpOpData::dcInstance

Instance to which the request is to be enqueued

Definition at line 112 of file cpa_dc_dp.h.

5.6.2.9 pSessionHandle

CpaDcSessionHandle _CpaDcDpOpData::pSessionHandle

DC Session associated with the stream of requests. This field is only valid when using the session based API functions. This field must be set to NULL if the application wishes to use the No-Session (Ns) API.

Definition at line 115 of file cpa_dc_dp.h.

5.6.2.10 srcBuffer

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.

Definition at line 122 of file cpa_dc_dp.h.

5.6.2.11 srcBufferLen

Cpa32U _CpaDcDpOpData::srcBufferLen

If the source buffer is a "flat buffer", then this field specifies the size of the buffer, in bytes. If the source buffer is a "buffer list" (of type CpaPhysBufferList), then this field should be set to the value CPA_DP_BUFLIST.

Definition at line 130 of file cpa_dc_dp.h.

5.6.2.12 destBuffer

```
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.

Definition at line 137 of file cpa_dc_dp.h.

5.6.2.13 destBufferLen

```
Cpa32U _CpaDcDpOpData::destBufferLen
```

If the destination buffer is a "flat buffer", then this field specifies the size of the buffer, in bytes. If the destination buffer is a "buffer list" (of type CpaPhysBufferList), then this field should be set to the value CPA_DP_BUFLIST.

Definition at line 145 of file cpa_dc_dp.h.

5.6.2.14 sessDirection

```
CpaDcSessionDir _CpaDcDpOpData::sessDirection
```

Session direction indicating whether session is used for compression, decompression. For the DP implementation, CPA_DC_DIR_COMBINED is not a valid selection.

Definition at line 152 of file cpa_dc_dp.h.

5.6.2.15 compressAndVerify

```
CpaBoolean _CpaDcDpOpData::compressAndVerify
```

If set to true, for compression operations, the implementation will verify that compressed data, generated by the compression operation, can be successfully decompressed. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerify capability.

Definition at line 158 of file cpa_dc_dp.h.

5.6.2.16 compressAndVerifyAndRecover

```
CpaBoolean _CpaDcDpOpData::compressAndVerifyAndRecover
```

If set to true, for compression operations, the implementation will automatically recover from a compressAndVerify error. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerifyAndRecover capability. The compressAndVerify field in CpaDcOpData MUST be set to CPA_TRUE if compressAndVerifyAndRecover is set to CPA_TRUE.

Definition at line 166 of file cpa_dc_dp.h.

5.6.2.17 responseStatus

```
CpaStatus _CpaDcDpOpData::responseStatus
```

Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Definition at line 175 of file cpa_dc_dp.h.

5.6.2.18 thisPhys

```
CpaPhysicalAddr _CpaDcDpOpData::thisPhys
```

Physical address of this data structure

Definition at line 180 of file cpa_dc_dp.h.

5.6.2.19 pCallbackTag

```
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.

Definition at line 183 of file cpa_dc_dp.h.

5.6.2.20 pSetupData

```
CpaDcNsSetupData* _CpaDcDpOpData::pSetupData
```

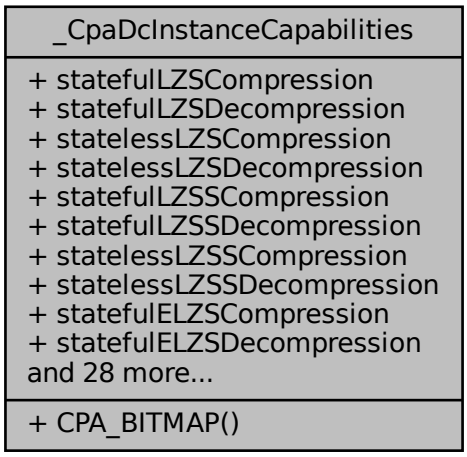
Pointer to the No-session (Ns) Setup data for configuration of this request.

This CpaDcNsSetupData structure must be initialised when using the Data Plane No-Session (Ns) API. Otherwise it should be set to NULL. When initialized, the existing Data Plane API functions can be used as is.

Definition at line 193 of file cpa_dc_dp.h.

5.7 _CpaDcInstanceCapabilities Struct Reference

Collaboration diagram for _CpaDcInstanceCapabilities:



Public Member Functions

- CPA_BITMAP (dcChainCapInfo, CPA_DC_CHAIN_CAP_BITMAP_SIZE)

Data Fields

- CpaBoolean statefulLZSCompression
- CpaBoolean statefulLZSDecompression
- CpaBoolean statelessLZSCompression
- CpaBoolean statelessLZSDecompression
- CpaBoolean statefulLZSSCompression
- CpaBoolean statefulLZSSDecompression
- CpaBoolean statelessLZSSCompression
- CpaBoolean statelessLZSSDecompression

- CpaBoolean statefulELZSCompression
- CpaBoolean statefulELZSDecompression
- CpaBoolean statelessELZSCompression
- CpaBoolean statelessELZSDecompression
- CpaBoolean statefulDeflateCompression
- CpaBoolean statefulDeflateDecompression
- CpaBoolean statelessDeflateCompression
- CpaBoolean statelessDeflateDecompression
- CpaBoolean statelessLZ4Compression
- CpaBoolean statelessLZ4Decompression
- CpaBoolean statefulLZ4Decompression
- CpaBoolean statelessLZ4SCompression
- CpaBoolean checksumCRC32
- CpaBoolean checksumAdler32
- CpaBoolean checksumXXHash32
- CpaBoolean dynamicHuffman
- CpaBoolean dynamicHuffmanBufferReq
- CpaBoolean precompiledHuffman
- CpaBoolean autoSelectBestHuffmanTree
- Cpa8U validWindowSizeMaskCompression
- Cpa8U validWindowSizeMaskDecompression
- Cpa32U internalHuffmanMem
- CpaBoolean endOfLastBlock
- CpaBoolean reportParityError
- CpaBoolean batchAndPack
- CpaBoolean compressAndVerify
- CpaBoolean compressAndVerifyStrict
- CpaBoolean compressAndVerifyAndRecover
- CpaBoolean integrityCrcs
- CpaBoolean integrityCrcs64b

5.7.1 Detailed Description

Implementation Capabilities Structure

Description:

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.

Definition at line 594 of file cpa_dc.h.

5.7.2 Member Function Documentation

5.7.2.1 CPA_BITMAP()

```
_CpaDcInstanceCapabilities::CPA_BITMAP (
            dcChainCapInfo ,
            CPA_DC_CHAIN_CAP_BITMAP_SIZE  )
```

Bitmap representing which chaining capabilities are supported by the instance. Bits can be tested using the macro CPA_BITMAP_BIT_TEST. The bit positions are those specified in the enumerated type CpaDcChainOperations in cpa_dc_chain.h.

5.7.3 Field Documentation

5.7.3.1 statefulLZSCompression

```
CpaBoolean _CpaDcInstanceCapabilities::statefulLZSCompression
```

True if the Instance supports Stateful LZS compression

Definition at line 595 of file cpa_dc.h.

5.7.3.2 statefulLZSDecompression

```
CpaBoolean _CpaDcInstanceCapabilities::statefulLZSDecompression
```

True if the Instance supports Stateful LZS decompression

Definition at line 597 of file cpa_dc.h.

5.7.3.3 statelessLZSCompression

```
CpaBoolean _CpaDcInstanceCapabilities::statelessLZSCompression
```

True if the Instance supports Stateless LZS compression

Definition at line 599 of file cpa_dc.h.

5.7.3.4 statelessLZSDecompression

```
CpaBoolean _CpaDcInstanceCapabilities::statelessLZSDecompression
```

True if the Instance supports Stateless LZS decompression

Definition at line 601 of file cpa_dc.h.

5.7.3.5 statefulLZSSCompression

`CpaBoolean _CpaDcInstanceCapabilities::statefulLZSSCompression`

True if the Instance supports Stateful LZSS compression

Definition at line 603 of file cpa_dc.h.

5.7.3.6 statefulLZSSDecompression

`CpaBoolean _CpaDcInstanceCapabilities::statefulLZSSDecompression`

True if the Instance supports Stateful LZSS decompression

Definition at line 605 of file cpa_dc.h.

5.7.3.7 statelessLZSSCompression

`CpaBoolean _CpaDcInstanceCapabilities::statelessLZSSCompression`

True if the Instance supports Stateless LZSS compression

Definition at line 607 of file cpa_dc.h.

5.7.3.8 statelessLZSSDecompression

`CpaBoolean _CpaDcInstanceCapabilities::statelessLZSSDecompression`

True if the Instance supports Stateless LZSS decompression

Definition at line 609 of file cpa_dc.h.

5.7.3.9 statefulELZSCompression

`CpaBoolean _CpaDcInstanceCapabilities::statefulELZSCompression`

True if the Instance supports Stateful Extended LZS compression

Definition at line 611 of file cpa_dc.h.

5.7.3.10 statefulELZSDecompression

CpaBoolean _CpaDcInstanceCapabilities::statefulELZSDecompression

True if the Instance supports Stateful Extended LZS decompression

Definition at line 614 of file cpa_dc.h.

5.7.3.11 statelessELZSCompression

CpaBoolean _CpaDcInstanceCapabilities::statelessELZSCompression

True if the Instance supports Stateless Extended LZS compression

Definition at line 617 of file cpa_dc.h.

5.7.3.12 statelessELZSDecompression

CpaBoolean _CpaDcInstanceCapabilities::statelessELZSDecompression

True if the Instance supports Stateless Extended LZS decompression

Definition at line 620 of file cpa_dc.h.

5.7.3.13 statefulDeflateCompression

CpaBoolean _CpaDcInstanceCapabilities::statefulDeflateCompression

True if the Instance supports Stateful Deflate compression

Definition at line 623 of file cpa_dc.h.

5.7.3.14 statefulDeflateDecompression

CpaBoolean _CpaDcInstanceCapabilities::statefulDeflateDecompression

True if the Instance supports Stateful Deflate decompression

Definition at line 625 of file cpa_dc.h.

5.7.3.15 statelessDeflateCompression

`CpaBoolean _CpaDcInstanceCapabilities::statelessDeflateCompression`

True if the Instance supports Stateless Deflate compression

Definition at line 628 of file cpa_dc.h.

5.7.3.16 statelessDeflateDecompression

`CpaBoolean _CpaDcInstanceCapabilities::statelessDeflateDecompression`

True if the Instance supports Stateless Deflate decompression

Definition at line 630 of file cpa_dc.h.

5.7.3.17 statelessLZ4Compression

`CpaBoolean _CpaDcInstanceCapabilities::statelessLZ4Compression`

True if the Instance supports Stateless LZ4 compression

Definition at line 633 of file cpa_dc.h.

5.7.3.18 statelessLZ4Decompression

`CpaBoolean _CpaDcInstanceCapabilities::statelessLZ4Decompression`

True if the Instance supports Stateless LZ4 decompression

Definition at line 635 of file cpa_dc.h.

5.7.3.19 statefulLZ4Decompression

`CpaBoolean _CpaDcInstanceCapabilities::statefulLZ4Decompression`

True if the Instance supports Stateful LZ4 decompression

Definition at line 637 of file cpa_dc.h.

5.7.3.20 statelessLZ4SCompression

```
CpaBoolean _CpaDcInstanceCapabilities::statelessLZ4SCompression
```

True if the Instance supports Stateless LZ4S compression

Definition at line 639 of file cpa_dc.h.

5.7.3.21 checksumCRC32

```
CpaBoolean _CpaDcInstanceCapabilities::checksumCRC32
```

True if the Instance can calculate a CRC32 checksum over the uncompressed data. This value is only calculated when CPA_DC_DEFLATE is configured as the algorithm for CpaDcCompType

Definition at line 641 of file cpa_dc.h.

5.7.3.22 checksumAdler32

```
CpaBoolean _CpaDcInstanceCapabilities::checksumAdler32
```

True if the Instance can calculate an Adler-32 checksum over the uncompressed data. This value is only calculated when CPA_DC_DEFLATE is configured as the algorithm for CpaDcCompType

Definition at line 646 of file cpa_dc.h.

5.7.3.23 checksumXXHash32

```
CpaBoolean _CpaDcInstanceCapabilities::checksumXXHash32
```

True if the Instance can calculate an xxHash-32 hash over the uncompressed data. This value is only calculated when CPA_DC_LZ4 or CPA_DC_LZ4S is configured as the algorithm for CpaDcCompType

Definition at line 651 of file cpa_dc.h.

5.7.3.24 dynamicHuffman

```
CpaBoolean _CpaDcInstanceCapabilities::dynamicHuffman
```

True if the Instance supports dynamic Huffman trees in deflate blocks

Definition at line 656 of file cpa_dc.h.

5.7.3.25 dynamicHuffmanBufferReq

CpaBoolean _CpaDcInstanceCapabilities::dynamicHuffmanBufferReq

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

Definition at line 659 of file cpa_dc.h.

5.7.3.26 precompiledHuffman

CpaBoolean _CpaDcInstanceCapabilities::precompiledHuffman

True if the Instance supports precompiled Huffman trees in deflate blocks

Definition at line 662 of file cpa_dc.h.

5.7.3.27 autoSelectBestHuffmanTree

CpaBoolean _CpaDcInstanceCapabilities::autoSelectBestHuffmanTree

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

Definition at line 665 of file cpa_dc.h.

5.7.3.28 validWindowSizeMaskCompression

Cpa8U _CpaDcInstanceCapabilities::validWindowSizeMaskCompression

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

Definition at line 669 of file cpa_dc.h.

5.7.3.29 validWindowSizeMaskDecompression

Cpa8U _CpaDcInstanceCapabilities::validWindowSizeMaskDecompression

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

Definition at line 672 of file cpa_dc.h.

5.7.3.30 internalHuffmanMem

```
Cpa32U _CpaDcInstanceCapabilities::internalHuffmanMem
```

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

Definition at line 675 of file cpa_dc.h.

5.7.3.31 endOfLastBlock

```
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.

Definition at line 678 of file cpa_dc.h.

5.7.3.32 reportParityError

```
CpaBoolean _CpaDcInstanceCapabilities::reportParityError
```

True if the instance supports parity error reporting.

Definition at line 683 of file cpa_dc.h.

5.7.3.33 batchAndPack

```
CpaBoolean _CpaDcInstanceCapabilities::batchAndPack
```

True if the instance supports 'batch and pack' compression

Definition at line 685 of file cpa_dc.h.

5.7.3.34 compressAndVerify

```
CpaBoolean _CpaDcInstanceCapabilities::compressAndVerify
```

True if the instance supports checking that compressed data, generated as part of a compression operation, can be successfully decompressed.

Definition at line 687 of file cpa_dc.h.

5.7.3.35 compressAndVerifyStrict

```
CpaBoolean _CpaDcInstanceCapabilities::compressAndVerifyStrict
```

True if compressAndVerify is 'strictly' enabled for the instance. If strictly enabled, compressAndVerify will be enabled by default for compression operations and cannot be disabled by setting opData.compressAndVerify=0 with cpaDcCompressData2(). Compression operations with opData.compressAndVerify=0 will return a CPA_STATUS_INVALID_PARAM error status when in compressAndVerify strict mode.

Definition at line 691 of file cpa_dc.h.

5.7.3.36 compressAndVerifyAndRecover

```
CpaBoolean _CpaDcInstanceCapabilities::compressAndVerifyAndRecover
```

True if the instance supports recovering from errors detected by compressAndVerify by generating a stored block in the compressed output data buffer. This stored block replaces any compressed content that resulted in a compressAndVerify error.

Definition at line 699 of file cpa_dc.h.

5.7.3.37 integrityCrcs

```
CpaBoolean _CpaDcInstanceCapabilities::integrityCrcs
```

True if the instance supports 32 bit integrity CRC checking in the compression/decompression datapath. Refer to CpaDcOpData for more details on integrity checking.

Definition at line 704 of file cpa_dc.h.

5.7.3.38 integrityCrcs64b

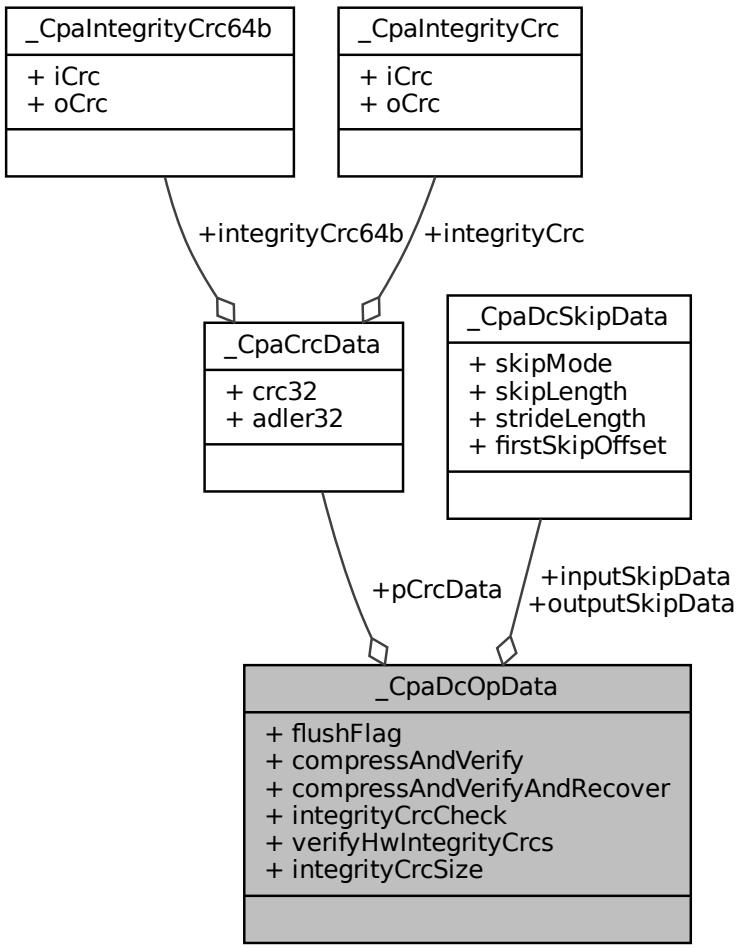
```
CpaBoolean _CpaDcInstanceCapabilities::integrityCrcs64b
```

True if the instance supports 64 bit integrity CRC checking in the compression / decompression datapath. Refer to CpaDcOpData for more details on integrity checking.

Definition at line 714 of file cpa_dc.h.

5.8 _CpaDcOpData Struct Reference

Collaboration diagram for _CpaDcOpData:



Data Fields

- CpaDcFlush flushFlag
- CpaBoolean compressAndVerify
- CpaBoolean compressAndVerifyAndRecover
- CpaBoolean integrityCrcCheck
- CpaBoolean verifyHwIntegrityCrcs
- CpaDcIntegrityCrcSize integrityCrcSize
- CpaDcSkipData inputSkipData
- CpaDcSkipData outputSkipData
- CpaCrcData * pCrcData

5.8.1 Detailed Description

(De)Compression request input parameters.

Description:

This structure contains the request information for use with compression operations.

Definition at line 1000 of file cpa_dc.h.

5.8.2 Field Documentation

5.8.2.1 flushFlag

`CpaDcFlush _CpaDcOpData::flushFlag`

Indicates the type of flush to be performed.

Definition at line 1001 of file cpa_dc.h.

5.8.2.2 compressAndVerify

`CpaBoolean _CpaDcOpData::compressAndVerify`

If set to true, for compression operations, the implementation will verify that compressed data, generated by the compression operation, can be successfully decompressed. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerify capability.

Definition at line 1003 of file cpa_dc.h.

5.8.2.3 compressAndVerifyAndRecover

`CpaBoolean _CpaDcOpData::compressAndVerifyAndRecover`

If set to true, for compression operations, the implementation will automatically recover from a compressAndVerify error. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerifyAndRecover capability. The compressAndVerify field in CpaDcOpData MUST be set to CPA_TRUE if compressAndVerifyAndRecover is set to CPA_TRUE.

Definition at line 1010 of file cpa_dc.h.

5.8.2.4 integrityCrcCheck

`CpaBoolean _CpaDcOpData::integrityCrcCheck`

If set to true, the implementation will verify that data integrity is preserved through the processing pipeline.

Integrity CRC checking is not supported for decompression operations over data that contains multiple gzip headers.

Definition at line 1018 of file cpa_dc.h.

5.8.2.5 verifyHwIntegrityCrcs

`CpaBoolean _CpaDcOpData::verifyHwIntegrityCrcs`

If set to true, software calculated CRCs will be compared against hardware generated integrity CRCs to ensure that data integrity is maintained when transferring data to and from the hardware accelerator.

Definition at line 1024 of file cpa_dc.h.

5.8.2.6 integrityCrcSize

`CpaDcIntegrityCrcSize _CpaDcOpData::integrityCrcSize`

This option specifies the size of the CRC to be used for data integrity checking. As such it is only valid if this request is configured for data integrity checks.

Definition at line 1029 of file cpa_dc.h.

5.8.2.7 inputSkipData

`CpaDcSkipData _CpaDcOpData::inputSkipData`

Optional skip regions in the input buffers

Definition at line 1033 of file cpa_dc.h.

5.8.2.8 outputSkipData

`CpaDcSkipData _CpaDcOpData::outputSkipData`

Optional skip regions in the output buffers

Definition at line 1035 of file cpa_dc.h.

5.8.2.9 pCrcData

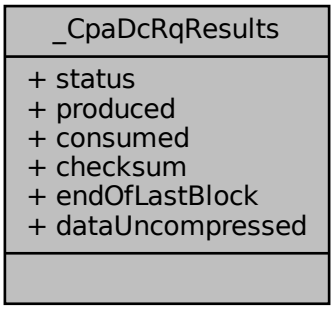
```
CpaCrcData* _CpaDcOpData::pCrcData
```

Pointer to CRCs for this operation, when integrity checks are enabled.

Definition at line 1037 of file cpa_dc.h.

5.9 _CpaDcRqResults Struct Reference

Collaboration diagram for _CpaDcRqResults:



Data Fields

- CpaDcReqStatus status
- Cpa32U produced
- Cpa32U consumed
- Cpa32U checksum
- CpaBoolean endOfLastBlock
- CpaBoolean dataUncompressed

5.9.1 Detailed Description

Request results data

Description:

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.

Definition at line 865 of file cpa_dc.h.

5.9.2 Field Documentation

5.9.2.1 status

`CpaDcReqStatus _CpaDcRqResults::status`

Additional status details from accelerator

Definition at line 866 of file cpa_dc.h.

5.9.2.2 produced

`Cpa32U _CpaDcRqResults::produced`

Octets produced by the operation

Definition at line 868 of file cpa_dc.h.

5.9.2.3 consumed

`Cpa32U _CpaDcRqResults::consumed`

Octets consumed by the operation

Definition at line 870 of file cpa_dc.h.

5.9.2.4 checksum

`Cpa32U _CpaDcRqResults::checksum`

The checksum produced by the operation. For some checksum algorithms, setting this field on the input to a stateless compression/decompression request can be used to pass in an initial checksum value that will be used to seed the checksums produced by the stateless operation.

The checksum algorithm CPA_DC_XXHASH32 does not support passing an input value in this parameter. Any initial value passed will be ignored by the compression/decompression operation when this checksum algorithm is used.

Definition at line 872 of file cpa_dc.h.

5.9.2.5 endOfLastBlock

```
CpaBoolean _CpaDcRqResults::endOfLastBlock
```

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

Definition at line 883 of file cpa_dc.h.

5.9.2.6 dataUncompressed

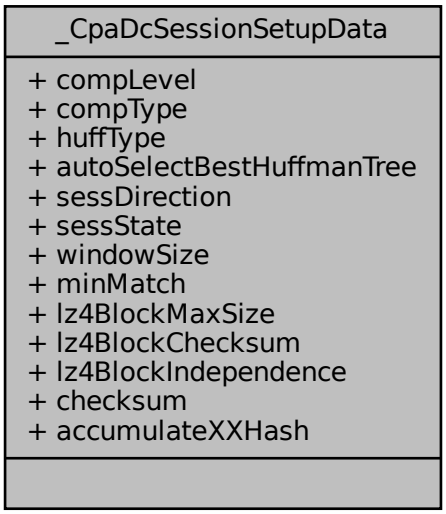
```
CpaBoolean _CpaDcRqResults::dataUncompressed
```

If TRUE the output data for this request is uncompressed and in the format setup for the request. This value is only valid for CPA_DC_ASB_ENABLED or if compressAndVerifyAndRecover is set to TRUE in the CpaDcOpData structure for a request.

Definition at line 886 of file cpa_dc.h.

5.10 _CpaDcSessionSetupData Struct Reference

Collaboration diagram for _CpaDcSessionSetupData:



Data Fields

- CpaDcCompLvl compLevel
- CpaDcCompType compType
- CpaDcHuffType huffType
- CpaDcAutoSelectBest autoSelectBestHuffmanTree
- CpaDcSessionDir sessDirection
- CpaDcSessionState sessState
- CpaDcCompWindowSize windowSize
- CpaDcCompMinMatch minMatch
- CpaDcCompLZ4BlockMaxSize lz4BlockMaxSize
- CpaBoolean lz4BlockChecksum
- CpaBoolean lz4BlockIndependence
- CpaDcChecksum checksum
- CpaBoolean accumulateXXHash

5.10.1 Detailed Description

Session Setup Data.

Description:

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.

Definition at line 730 of file cpa_dc.h.

5.10.2 Field Documentation

5.10.2.1 compLevel

```
CpaDcCompLvl _CpaDcSessionSetupData::compLevel
```

Compression Level from CpaDcCompLvl

Definition at line 731 of file cpa_dc.h.

5.10.2.2 compType

```
CpaDcCompType _CpaDcSessionSetupData::compType
```

Compression type from CpaDcCompType

Definition at line 733 of file cpa_dc.h.

5.10.2.3 huffType

CpaDcHuffType _CpaDcSessionSetupData::huffType

Huffman type from CpaDcHuffType

Definition at line 735 of file cpa_dc.h.

5.10.2.4 autoSelectBestHuffmanTree

CpaDcAutoSelectBest _CpaDcSessionSetupData::autoSelectBestHuffmanTree

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

Definition at line 737 of file cpa_dc.h.

5.10.2.5 sessDirection

CpaDcSessionDir _CpaDcSessionSetupData::sessDirection

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

Definition at line 740 of file cpa_dc.h.

5.10.2.6 sessState

CpaDcSessionState _CpaDcSessionSetupData::sessState

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

Definition at line 743 of file cpa_dc.h.

5.10.2.7 windowSize

CpaDcCompWindowSize _CpaDcSessionSetupData::windowSize

Window size from CpaDcCompWindowSize

Definition at line 746 of file cpa_dc.h.

5.10.2.8 minMatch

```
CpaDcCompMinMatch _CpaDcSessionSetupData::minMatch
```

Min Match size from CpaDcCompMinMatch

Definition at line 748 of file cpa_dc.h.

5.10.2.9 lz4BlockMaxSize

```
CpaDcCompLZ4BlockMaxSize _CpaDcSessionSetupData::lz4BlockMaxSize
```

Window size from CpaDcCompLZ4BlockMaxSize

Definition at line 750 of file cpa_dc.h.

5.10.2.10 lz4BlockChecksum

```
CpaBoolean _CpaDcSessionSetupData::lz4BlockChecksum
```

LZ4 Block Checksum setting for the LZ4 request. For LZ4 decompression operations, this setting must be set based on the B.Checksum flag originating from the LZ4 frame header. For LZ4 compression operations, this setting will be ignored as the implementation does not support generation of Data Block checksums.

Definition at line 752 of file cpa_dc.h.

5.10.2.11 lz4BlockIndependence

```
CpaBoolean _CpaDcSessionSetupData::lz4BlockIndependence
```

LZ4 Block Independence Flag setting. For LZ4 compression operations, this setting must be set based on the Block Independence Flag originating from the LZ4 frame header. For LZ4 decompression operations, this setting is ignored. For data compressed with lz4BlockIndependence set to CPA_FALSE, it is not possible to perform parallel decompression on the compressed blocks. It is also not possible to access the produced LZ4 blocks randomly.

Definition at line 759 of file cpa_dc.h.

5.10.2.12 checksum

```
CpaDcChecksum _CpaDcSessionSetupData::checksum
```

Desired checksum required for the session

Definition at line 769 of file cpa_dc.h.

5.10.2.13 accumulateXXHash

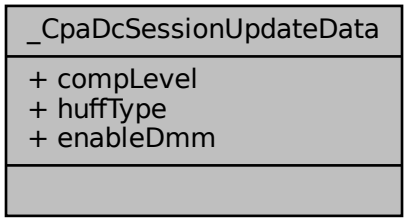
```
CpaBoolean _CpaDcSessionSetupData::accumulateXXHash
```

If TRUE the xxHash calculation for LZ4 requests using the session based API, cpaDcCompressData2 and cpaDcCompressData, will be accumulated across requests, with a valid xxHash being written to CpaDcRqResults.checksum for the request which specifies CPA_DC_FLUSH_FINAL in CpaDcOpData.flushFlag. When the CPA_DC_FLUSH_FINAL is received, the internal XXHash state will be reset for this session. One exception is if a CPA_DC_OVERFLOW error is returned, the xxHash value in the checksum field will be valid for requests up to that point and the internal XXHash state will not be reset. This will allow a user to either create an LZ4 frame based off the data at the time of overflow, or correct the overflow condition and continue submitting requests until specifying CPA_DC_FLUSH_FINAL. Additionally the user can force the internal XXHash state to reset (even on overflow) by calling cpaDcResetXXHashState on this session. For the sessionless API, cpaDcNsCompressData, this flag will have no effect

Definition at line 771 of file cpa_dc.h.

5.11 _CpaDcSessionUpdateData Struct Reference

Collaboration diagram for _CpaDcSessionUpdateData:



Data Fields

- CpaDcCompLvl compLevel
- CpaDcHuffType huffType
- CpaBoolean enableDmm

5.11.1 Detailed Description

Session Update Data.

Description:

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

Definition at line 803 of file cpa_dc.h.

5.11.2 Field Documentation

5.11.2.1 compLevel

`CpaDcCompLvl _CpaDcSessionUpdateData::compLevel`

Compression Level from CpaDcCompLvl

Definition at line 804 of file cpa_dc.h.

5.11.2.2 huffType

`CpaDcHuffType _CpaDcSessionUpdateData::huffType`

Huffman type from CpaDcHuffType

Definition at line 806 of file cpa_dc.h.

5.11.2.3 enableDmm

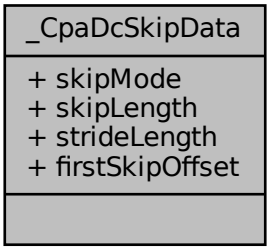
`CpaBoolean _CpaDcSessionUpdateData::enableDmm`

Desired DMM required for the session

Definition at line 808 of file cpa_dc.h.

5.12 _CpaDcSkipData Struct Reference

Collaboration diagram for _CpaDcSkipData:



Data Fields

- CpaDcSkipMode skipMode
- Cpa32U skipLength
- Cpa32U strideLength
- Cpa32U firstSkipOffset

5.12.1 Detailed Description

Skip Region Data.

Description:

This structure contains data relating to configuring skip region behaviour. A skip region is a region of an input buffer that should be omitted from processing or a region that should be inserted into the output buffer.

Definition at line 978 of file cpa_dc.h.

5.12.2 Field Documentation

5.12.2.1 skipMode

```
CpaDcSkipMode _CpaDcSkipData::skipMode
```

Skip mode from CpaDcSkipMode for buffer processing

Definition at line 979 of file cpa_dc.h.

5.12.2.2 skipLength

```
Cpa32U _CpaDcSkipData::skipLength
```

Number of bytes to skip when skip mode is enabled

Definition at line 981 of file cpa_dc.h.

5.12.2.3 strideLength

```
Cpa32U _CpaDcSkipData::strideLength
```

Size of the stride between skip regions when skip mode is set to CPA_DC_SKIP_STRIDE.

Definition at line 983 of file cpa_dc.h.

5.12.2.4 firstSkipOffset

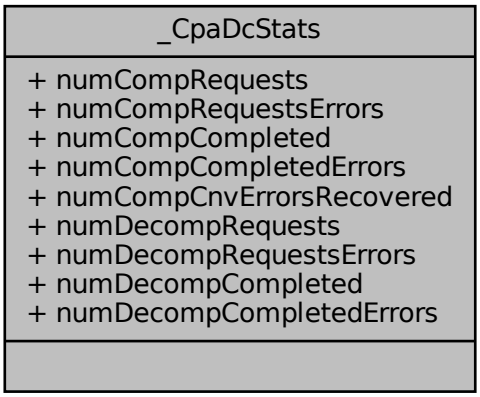
```
Cpa32U _CpaDcSkipData::firstSkipOffset
```

Number of bytes to skip in a buffer before reading/writing the input/output data.

Definition at line 986 of file cpa_dc.h.

5.13 _CpaDcStats Struct Reference

Collaboration diagram for _CpaDcStats:



Data Fields

- Cpa64U numCompRequests
- Cpa64U numCompRequestsErrors
- Cpa64U numCompCompleted
- Cpa64U numCompCompletedErrors
- Cpa64U numCompCnvErrorsRecovered
- Cpa64U numDecompRequests
- Cpa64U numDecompRequestsErrors
- Cpa64U numDecompCompleted
- Cpa64U numDecompCompletedErrors

5.13.1 Detailed Description

Compression Statistics Data.

Description:

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.

Definition at line 822 of file cpa_dc.h.

5.13.2 Field Documentation

5.13.2.1 numCompRequests

Cpa64U _CpaDcStats::numCompRequests

Number of successful compression requests

Definition at line 823 of file cpa_dc.h.

5.13.2.2 numCompRequestsErrors

Cpa64U _CpaDcStats::numCompRequestsErrors

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

Definition at line 825 of file cpa_dc.h.

5.13.2.3 numCompCompleted

Cpa64U _CpaDcStats::numCompCompleted

Compression requests completed

Definition at line 828 of file cpa_dc.h.

5.13.2.4 numCompCompletedErrors

Cpa64U _CpaDcStats::numCompCompletedErrors

Compression requests not completed due to errors

Definition at line 830 of file cpa_dc.h.

5.13.2.5 numCompCnvErrorsRecovered

Cpa64U _CpaDcStats::numCompCnvErrorsRecovered

Compression CNV errors that have been recovered

Definition at line 832 of file cpa_dc.h.

5.13.2.6 numDecompRequests

`Cpa64U _CpaDcStats::numDecompRequests`

Number of successful decompression requests

Definition at line 835 of file cpa_dc.h.

5.13.2.7 numDecompRequestsErrors

`Cpa64U _CpaDcStats::numDecompRequestsErrors`

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

Definition at line 837 of file cpa_dc.h.

5.13.2.8 numDecompCompleted

`Cpa64U _CpaDcStats::numDecompCompleted`

Decompression requests completed

Definition at line 840 of file cpa_dc.h.

5.13.2.9 numDecompCompletedErrors

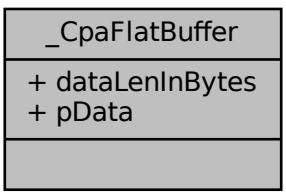
`Cpa64U _CpaDcStats::numDecompCompletedErrors`

Decompression requests not completed due to errors

Definition at line 842 of file cpa_dc.h.

5.14 _CpaFlatBuffer Struct Reference

Collaboration diagram for _CpaFlatBuffer:



Data Fields

- Cpa32U dataLenInBytes
- Cpa8U * pData

5.14.1 Detailed Description

Flat buffer structure containing a pointer and length member.

Description:

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.

Definition at line 129 of file cpa.h.

5.14.2 Field Documentation

5.14.2.1 dataLenInBytes

```
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.

Definition at line 130 of file cpa.h.

5.14.2.2 pData

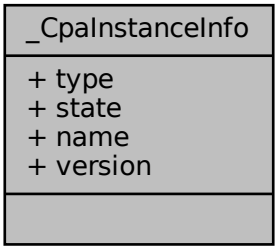
```
Cpa8U* _CpaFlatBuffer::pData
```

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.

Definition at line 138 of file cpa.h.

5.15 _CpaInstanceInfo Struct Reference

Collaboration diagram for _CpaInstanceInfo:



Data Fields

- enum _CpaInstanceType type
- enum _CpaInstanceState state
- Cpa8U name [CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]
- Cpa8U version [CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES]

5.15.1 Detailed Description

Instance Info Structure

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

Description:

Structure that contains the information to describe the instance.

Definition at line 463 of file cpa.h.

5.15.2 Field Documentation

5.15.2.1 type

```
enum _CpaInstanceType _CpaInstanceInfo::type
```

Type definition for this instance.

Definition at line 464 of file cpa.h.

5.15.2.2 state

```
enum _CpaInstanceState _CpaInstanceInfo::state
```

Operational state of the instance.

Definition at line 466 of file cpa.h.

5.15.2.3 name

```
Cpa8U _CpaInstanceInfo::name[CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]
```

Simple text string identifier for the instance.

Definition at line 468 of file cpa.h.

5.15.2.4 version

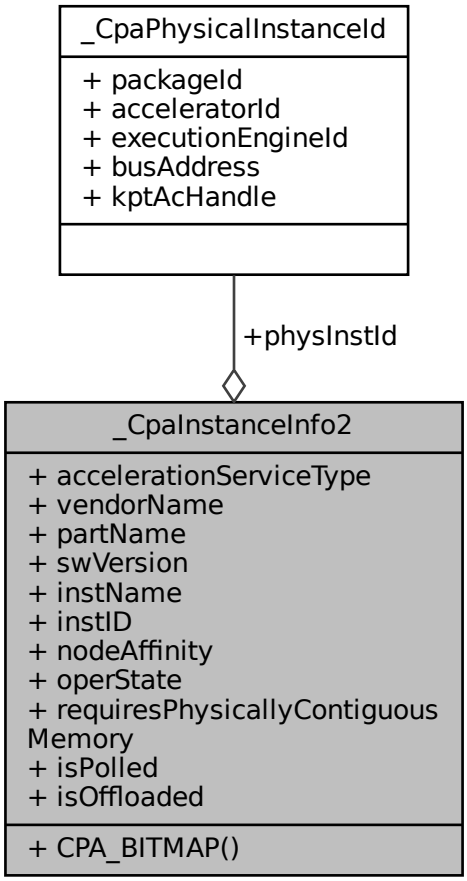
```
Cpa8U _CpaInstanceInfo::version[CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES]
```

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

Definition at line 470 of file cpa.h.

5.16 _CpaInstanceInfo2 Struct Reference

Collaboration diagram for _CpaInstanceInfo2:



Public Member Functions

- CPA_BITMAP (coreAffinity, CPA_MAX_CORES)

Data Fields

- CpaAccelerationServiceType accelerationServiceType
- Cpa8U vendorName [CPA_INST_VENDOR_NAME_SIZE]
- Cpa8U partName [CPA_INST_PART_NAME_SIZE]
- Cpa8U swVersion [CPA_INST_SW_VERSION_SIZE]
- Cpa8U instName [CPA_INST_NAME_SIZE]
- Cpa8U instID [CPA_INST_ID_SIZE]
- CpaPhysicalInstanceId physInstId
- Cpa32U nodeAffinity
- CpaOperationalState operState
- CpaBoolean requiresPhysicallyContiguousMemory
- CpaBoolean isPolled
- CpaBoolean isOffloaded

5.16.1 Detailed Description

Instance Info Structure, version 2

Description:

Structure that contains the information to describe the instance.

Definition at line 525 of file cpa.h.

5.16.2 Member Function Documentation

5.16.2.1 CPA_BITMAP()

```
_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.

5.16.3 Field Documentation

5.16.3.1 accelerationServiceType

```
CpaAccelerationServiceType _CpaInstanceInfo2::accelerationServiceType
```

Type of service provided by this instance.

Definition at line 526 of file cpa.h.

5.16.3.2 vendorName

```
Cpa8U _CpaInstanceInfo2::vendorName[CPA_INST_VENDOR_NAME_SIZE]
```

String identifying the vendor of the accelerator.

Definition at line 530 of file cpa.h.

5.16.3.3 partName

```
Cpa8U _CpaInstanceInfo2::partName[CPA_INST_PART_NAME_SIZE]
```

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

Definition at line 535 of file cpa.h.

5.16.3.4 swVersion

```
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).

Definition at line 540 of file cpa.h.

5.16.3.5 instName

```
Cpa8U _CpaInstanceInfo2::instName[CPA_INST_NAME_SIZE]
```

String identifying the name of the instance.

Definition at line 553 of file cpa.h.

5.16.3.6 instID

```
Cpa8U _CpaInstanceInfo2::instID[CPA_INST_ID_SIZE]
```

String containing a unique identifier for the instance

Definition at line 557 of file cpa.h.

5.16.3.7 physInstId

```
CpaPhysicalInstanceId _CpaInstanceInfo2::physInstId
```

Identifies the "physical instance" of the accelerator.

Definition at line 560 of file cpa.h.

5.16.3.8 nodeAffinity

```
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.

Definition at line 589 of file cpa.h.

5.16.3.9 operState

```
CpaOperationalState _CpaInstanceInfo2::operState
```

Operational state of the instance.

Definition at line 598 of file cpa.h.

5.16.3.10 requiresPhysicallyContiguousMemory

```
CpaBoolean _CpaInstanceInfo2::requiresPhysicallyContiguousMemory
```

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

Definition at line 600 of file cpa.h.

5.16.3.11 isPolled

```
CpaBoolean _CpaInstanceInfo2::isPolled
```

Specifies whether the instance must be polled, or is event driven. For hardware accelerators, the alternative to polling would be interrupts.

Definition at line 604 of file cpa.h.

5.16.3.12 isOffloaded

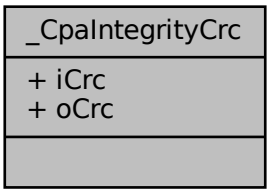
`CpaBoolean _CpaInstanceInfo2::isOffloaded`

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

Definition at line 608 of file cpa.h.

5.17 _CpaIntegrityCrc Struct Reference

Collaboration diagram for _CpaIntegrityCrc:



Data Fields

- Cpa32U iCrc
- Cpa32U oCrc

5.17.1 Detailed Description

Integrity CRC calculation details

Description:

This structure contains information about resulting integrity CRC calculations performed for a single request.

Definition at line 918 of file cpa_dc.h.

5.17.2 Field Documentation

5.17.2.1 iCrc

`Cpa32U _CpaIntegrityCrc::iCrc`

CRC calculated on request's input buffer

Definition at line 919 of file cpa_dc.h.

5.17.2.2 oCrc

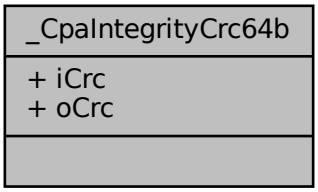
`Cpa32U _CpaIntegrityCrc::oCrc`

CRC calculated on request's output buffer

Definition at line 920 of file cpa_dc.h.

5.18 _CpaIntegrityCrc64b Struct Reference

Collaboration diagram for _CpaIntegrityCrc64b:



Data Fields

- Cpa64U iCrc
- Cpa64U oCrc

5.18.1 Detailed Description

Integrity CRC64 calculation details

Description:

This structure contains information about resulting integrity CRC64 calculations performed for a single request.

Definition at line 932 of file cpa_dc.h.

5.18.2 Field Documentation

5.18.2.1 iCrc

```
Cpa64U _CpaIntegrityCrc64b::iCrc
```

CRC calculated on request's input buffer

Definition at line 933 of file cpa_dc.h.

5.18.2.2 oCrc

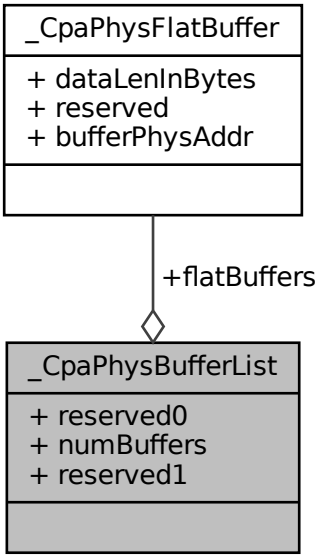
```
Cpa64U _CpaIntegrityCrc64b::oCrc
```

CRC calculated on request's output buffer

Definition at line 934 of file cpa_dc.h.

5.19 _CpaPhysBufferList Struct Reference

Collaboration diagram for _CpaPhysBufferList:



Data Fields

- Cpa64U reserved0
- Cpa32U numBuffers
- Cpa32U reserved1
- CpaPhysFlatBuffer flatBuffers [ ]

5.19.1 Detailed Description

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

Description:

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.

Definition at line 224 of file cpa.h.

5.19.2 Field Documentation

5.19.2.1 reserved0

```
Cpa64U _CpaPhysBufferList::reserved0
```

Reserved for internal usage

Definition at line 225 of file cpa.h.

5.19.2.2 numBuffers

```
Cpa32U _CpaPhysBufferList::numBuffers
```

Number of buffers in the list

Definition at line 227 of file cpa.h.

5.19.2.3 reserved1

```
Cpa32U _CpaPhysBufferList::reserved1
```

Reserved for alignment

Definition at line 229 of file cpa.h.

5.19.2.4 flatBuffers

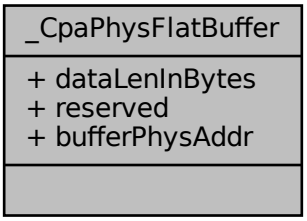
```
CpaPhysFlatBuffer _CpaPhysBufferList::flatBuffers[]
```

Array of flat buffer structures, of size numBuffers

Definition at line 231 of file cpa.h.

5.20 _CpaPhysFlatBuffer Struct Reference

Collaboration diagram for _CpaPhysFlatBuffer:



Data Fields

- Cpa32U dataLenInBytes
- Cpa32U reserved
- CpaPhysicalAddr bufferPhysAddr

5.20.1 Detailed Description

Flat buffer structure with physical address.

Description:

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

Definition at line 192 of file cpa.h.

5.20.2 Field Documentation

5.20.2.1 dataLenInBytes

`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.

Definition at line 193 of file cpa.h.

5.20.2.2 reserved

`Cpa32U _CpaPhysFlatBuffer::reserved`

Reserved for alignment

Definition at line 202 of file cpa.h.

5.20.2.3 bufferPhysAddr

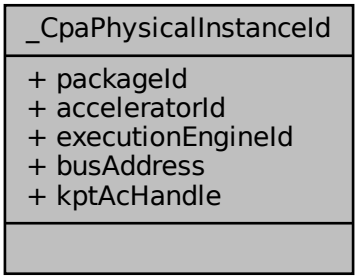
`CpaPhysicalAddr _CpaPhysFlatBuffer::bufferPhysAddr`

The physical address at which the data resides. The data pointed to is required to be in contiguous physical memory.

Definition at line 204 of file cpa.h.

5.21 _CpaPhysicalInstanceId Struct Reference

Collaboration diagram for _CpaPhysicalInstanceId:



Data Fields

- Cpa16U packageId
- Cpa16U acceleratorId
- Cpa16U executionEngineId
- Cpa16U busAddress
- Cpa32U kptAcHandle

5.21.1 Detailed Description

Physical Instance ID

Description:

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.

Definition at line 501 of file cpa.h.

5.21.2 Field Documentation

5.21.2.1 packageId

```
Cpa16U _CpaPhysicalInstanceId::packageId
```

Identifies the package within which the accelerator is contained.

Definition at line 502 of file cpa.h.

5.21.2.2 acceleratorId

```
Cpa16U _CpaPhysicalInstanceId::acceleratorId
```

Identifies the specific accelerator within the package.

Definition at line 505 of file cpa.h.

5.21.2.3 executionEngineId

```
Cpa16U _CpaPhysicalInstanceId::executionEngineId
```

Identifies the specific execution engine within the accelerator.

Definition at line 507 of file cpa.h.

5.21.2.4 busAddress

```
Cpa16U _CpaPhysicalInstanceId::busAddress
```

Identifies the bus address associated with the accelerator execution engine.

Definition at line 510 of file cpa.h.

5.21.2.5 kptAcHandle

```
Cpa32U _CpaPhysicalInstanceId::kptAcHandle
```

Identifies the achandle of the accelerator.

Definition at line 513 of file cpa.h.