Intel® QuickAssist Technology
Compression API

Reference

API Reference Guide

Automatically generated from sources, July 14, 2021.
Based on API version 2.7
(See Release Notes to map API version to software package version.)
Contents

1  Deprecated List .......................................................................................................................................................... 10
2  CPA API ....................................................................................................................................................................... 11
  2.1  DetailedDescription .................................................................................................................................................. 11
  2.2  Modules .................................................................................................................................................................... 11
3  Base Data Types .......................................................................................................................................................... 12
  3.1  DetailedDescription .................................................................................................................................................. 12
  3.2  Data Structures ........................................................................................................................................................ 12
  3.3  Defines ...................................................................................................................................................................... 12
  3.4  Typedefs .............................................................................................................................................................. 12
  3.5  Enumerations ....................................................................................................................................................... 13
  3.6  Data Structure Documentation ................................................................................................................................ 14
      3.6.1  _CpaFlatBuffer Struct Reference .................................................................................................................... 14
      3.6.2  _CpaBufferList Struct Reference ................................................................................................................ 15
      3.6.3  _CpaPhysFlatBuffer Struct Reference .......................................................................................................... 16
      3.6.4  _CpaPhysBufferList Struct Reference .......................................................................................................... 17
      3.6.5  _CpalnstanceInfo Struct Reference ........................................................................................................... 18
      3.6.6  _CpaPhysicalInstanceid Struct Reference ................................................................................................. 19
      3.6.7  _CpalnstanceInfo2 Struct Reference ........................................................................................................... 20
  3.7  Define Documentation ............................................................................................................................................ 22
  3.8  Typedef Documentation ........................................................................................................................................ 24
  3.9  Enumeration Type Documentation .................................................................................................................... 28
4  CPA Type Definition [CPA API] .................................................................................................................................. 30
  4.1  DetailedDescription ................................................................................................................................................ 30
  4.2  Defines .................................................................................................................................................................... 30
  4.3  Typedefs .............................................................................................................................................................. 30
  4.4  Enumerations ....................................................................................................................................................... 31
  4.5  Define Documentation ........................................................................................................................................... 31
  4.6  Typedef Documentation ........................................................................................................................................ 32
  4.7  Enumeration Type Documentation .................................................................................................................... 33
5  Data Compression API ............................................................................................................................................... 34
  5.1  DetailedDescription ................................................................................................................................................ 34
  5.2  Modules .................................................................................................................................................................... 34
  5.3  Data Structures ........................................................................................................................................................ 34
  5.4  Defines ................................................................................................................................................................... 35
  5.5  Typedefs .............................................................................................................................................................. 35
  5.6  Enumerations ....................................................................................................................................................... 36
  5.7  Functions ............................................................................................................................................................. 38
  5.8  Data Structure Documentation ................................................................................................................................ 40
      5.8.1  _CpaDcInstanceCapabilities Struct Reference ......................................................................................... 40
5.8.2 _CpaDcSessionSetupData Struct Reference.................................................. 44
5.8.3 _CpaDcSessionUpdateData Struct Reference................................................. 45
5.8.4 _CpaDcStats Struct Reference...................................................................... 46
5.8.5 _CpaDcRqResults Struct Reference............................................................... 47
5.8.6 _CpIntegrityCrc Struct Reference................................................................. 48
5.8.7 _CpaCrcData Struct Reference...................................................................... 49
5.8.8 _CpaDcSkipData Struct Reference................................................................. 50
5.8.9 _CpaDcOpData Struct Reference.................................................................. 51
5.9 Define Documentation...................................................................................... 53
5.10 Typedef Documentation.................................................................................. 53
5.11 Enumeration Type Documentation................................................................. 59
5.12 Function Documentation................................................................................ 64

6 Data Compression Batch and Pack API [Data Compression API].......................... 107
6.1 DetailedDescription........................................................................................ 107
6.2 Data Structures............................................................................................... 107
6.3 Typedefs......................................................................................................... 107
6.4 Functions........................................................................................................ 107
6.5 Data Structure Documentation........................................................................ 108
6.5.1 _CpaDcBatchOpData Struct Reference......................................................... 108
6.5.1.1 Detailed Description.................................................................................. 108
6.5.1.2 Data Fields.............................................................................................. 109
6.5.1.3 Field Documentation................................................................................ 109
6.6 Typedef Documentation.................................................................................. 109
6.7 Function Documentation................................................................................ 109

7 Data Compression Chaining API [Data Compression API].................................. 115
7.1 DetailedDescription........................................................................................ 115
7.2 Data Structures............................................................................................... 115
7.3 Typedefs......................................................................................................... 115
7.4 Enumerations................................................................................................ 115
7.5 Functions........................................................................................................ 116
7.6 Data Structure Documentation........................................................................ 117
7.6.1 _CpaDcChainSessionSetupData Struct Reference....................................... 117
7.6.1.1 Detailed Description.................................................................................. 117
7.6.1.2 Data Fields.............................................................................................. 117
7.6.1.3 Field Documentation................................................................................ 117
7.6.2 _CpaDcChainOpData Struct Reference......................................................... 118
7.6.2.1 Detailed Description.................................................................................. 119
7.6.2.2 Data Fields.............................................................................................. 119
7.6.2.3 Field Documentation................................................................................ 119
7.6.3 _CpaDcChainRqResults Struct Reference..................................................... 119
7.6.3.1 Detailed Description.................................................................................. 119
7.6.3.2 Data Fields.............................................................................................. 119
7.6.3.3 Field Documentation................................................................................ 120
7.7 Typedef Documentation.................................................................................. 120
7.8 Enumeration Type Documentation................................................................. 121
7.9 Function Documentation................................................................................ 123
8 Data Compression Data Plane API[Data Compression API] .......................................................... 132
8.1 DetailedDescription ...................................................................................................................... 132
8.2 Data Structures .......................................................................................................................... 133
8.3 Typedefs ......................................................................................................................................... 133
8.4 Functions ....................................................................................................................................... 133
8.5 Data Structure Documentation .................................................................................................. 134
  8.5.1 _CpaDcDpOpData Struct Reference ......................................................................................... 134
    8.5.1.1 Detailed Description ........................................................................................................ 134
    8.5.1.2 Data Fields ...................................................................................................................... 135
    8.5.1.3 Field Documentation ...................................................................................................... 135
8.6 Typedef Documentation ................................................................................................................ 137
8.7 Function Documentation .............................................................................................................. 138

Figures

Figure 1. Collaboration diagram for CPA API .................................................................................. 11
Figure 2. Collaboration diagram for Base Data Types: .................................................................. 12
Figure 3. Collaboration diagram for _CpaBufferList: ......................................................................... 15
Figure 4. Collaboration diagram for _CpaPhysBufferList ................................................................ 17
Figure 5. Collaboration diagram for _CpaInstanceInfo2 ................................................................... 20
Figure 6. Collaboration diagram for CPA Type Definition: ............................................................... 30
Figure 7. Collaboration diagram for Data Compression API: .............................................................. 34
Figure 8. Collaboration diagram for _CpaCrcData ............................................................................. 49
Figure 9. Collaboration diagram for _CpaDcOpData ........................................................................ 51
Figure 10. Collaboration diagram for Data Compression Batch and Pack API: ................................... 107
Figure 11. Collaboration diagram for _CpaDcBatchOpData ................................................................. 108
Figure 12. Collaboration diagram for Data Compression Chaining API ............................................ 115
Figure 13. Collaboration diagram for _CpaDcChainSessionSetupData ........................................... 117
Figure 14. Collaboration diagram for _CpaDcChainOpData: ............................................................... 118
Figure 15. Collaboration diagram for Data Compression Data Plane API: ........................................ 132
Figure 16. Collaboration diagram for _CpaDcDpOpData ................................................................. 134
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
</table>
| January 2022 | 014      | • Updated with new Intel® logo  
• Updated content with Intel One text font |
| July 2021    | 013      | • Compression API v2.7  
• Added CPA_DC_ASB_ENABLED  
• Added a flag to notify of an uncompressed block in CpaDcRqResults. |
| November 2020| 012      | • Changing version of the compression API to v2.6  
• Adding support for integrity CRCs. |
| September 2020| 011    | • Adding cpaDcDeflateCompressBound()  
• Changed version of the compression API to v2.5 |
| June 2020    | 010      | • Adding cpaDcUpdateSession()  
• Changed version of the compression API to v2.4 |
| January 2019 | 009      | • Adding chaining support.  
• Changed version of the compression API to v2.3 |
| April 2018   | 008      | • Adding support for compressAndVerifyAndRecover. |
| February 2018| 007      | • Adding support for Compress and Verify strict mode. |
| June 2016    | 006      | • Adding support for:  
• Compress and Verify  
• Batch and Pack Compression |
| October 2015 | 005      | • Changed version of the Compression API to v2.0.  
• Added new error codes to CpaDcReqStatus in cpa_dc.h. |
| September 2015| 004     | • Incrementing DC API version number to v1.6.  
• Adding CPA_STATUS_UNSUPPORTED as a return status for each function and callback  
• Deprecating use of fileType and deflateWindowSize fields from CpaDcSessionSetupData in cpa_dc.h.  
• Clarifying documentation for srcBufferLen, bufferLenToCompress, destBufferLen and bufferLenForData fields in CpaDcDpOpData in cpa_dc_dp.h  
• IXA00391973: Adding reportParityError to the DC instance capabilities. |
<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 2015</td>
<td>003</td>
<td>• Incrementing the DC API version number of 1.5</td>
</tr>
<tr>
<td>October 2014</td>
<td>002</td>
<td>• Adding cpaDcResetSession API</td>
</tr>
<tr>
<td>June 2014</td>
<td>001</td>
<td>• First public version of the document. Based on Intel Confidential document number 410926-1.3 with the revision history of that document retained for reference purposes.</td>
</tr>
</tbody>
</table>
| February 2013   | 1.3      | • Supports supplying multiple intermediate buffer lists when starting a compression instance. Also provides a utility function to determine the number of intermediate buffer lists required by an implementation.  
  • API extensions to support endOfLastBlock detection within a deflate stream. |
| January 2013    | 1.2      | • Resolves the following work requests:  
  • TECG00000185: Changing use of flush flags for stateless compression. Adding support for passing an initial checksum into a stateless compression request. Adding a constraint that cpaDcGenerateFooter() is not supported for stateless operations. |
| November 2012   | 1.1      | • Resolves the following work requests:  
  • TECG00000189: Add a unique instance identifier to CpaInstanceInfo2  
  • TECG00000193: Enhanced auto select best |
| October 2012    | 1.0      | • Resolves the following work requests:  
  • TECG00000186: Add instance notification support for RESTARTING & RESTARTED events and CPA_STATUS_RESTARTING return codes. |
| June 2012       | 0.93     | • Resolved review comments against previous version that resulted in minor updates to the API comments.  
  • Resolved the following work requests:  
  • TECG00000179: Adding version number to compression API |
| May 2012        | 0.92     | • Resolved the following work requests:  
  • TECG00000172: Remove references to cpaDcSessionCreate in cpa_dc.h  
  • TECG00000170: cpaCySymDpSessionCtxGetSize() returns a fixed value  
  • TECG00000173 and TECG00000174 updates/cleanup of API comments  
  • TECG00000174: Updated checksum processing rules. |
<p>| March 2012      | 0.92RC7  | • TECG00000166: Added ability to query bus address information for a CpaInstance. |
| November 2011   | 0.92RC5  | • Added internal memory store to capabilities reporting                     |
| September 2011  | 0.92RC4  | • Addressed review comments                                                 |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
</table>
| July 2011  | 0.92RC3  | • Completed data plane API  
• Moved results structure to first 64 byte section  
• Added buffer sizes for use by driver |
| May 2011   | 0.92RC2  | • Addressed comments in data plane API  |
| March 2011 | 0.92RC1  | • Added data plane API  |
| October 2010 | 0.91RC2 | • Minor typo fixes  |
| September 2010 | 0.91RC1 | • Based on feedback, incorporated the following:  
• Converted statistics counters to 64 bit  
• Improved the results structure  
• Updated memory configuration for consistency with other services |
| March 2010 | 0.9RC5   | • Based on review and feedback, incorporated the following:  
• 1. Added a results structure to the compress and decompress interfaces, and to the callback API  
• added enums to define the potential failures of the accelerators  
• Intermediate buffer is now a buffer list.  |
| January 2010 | 0.9RC4-2 | • Added size of context field to cpaDcGetSessionSize  |
| December 2009 | 0.90RC4 | • Based on feedback, incorporated the following:  
• Separated checksum algorithms in capabilities  
• Added return code CPA_DC_BAD_DATA return code  
• Bundled return information to include bytes consumed, bytes produced and checksum  
• Clean up of some documentation  |
| September 2009 | 0.90RC3 | • Updated as a result of review, incorporate the following changes.  
• Compression window capabilities now split for compress and decompress.  
• Update statistic to be more consistent with other APIs.  
• Added pHistoryBuffer to support stateful deflate.  
• Removed reference to having different instances able to process the same session.  |
<p>| July 2009   | 0.90RC2  | • Added distinction in capabilities for stateful and stateless, compression, and decompression  |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2009</td>
<td>0.90RC1</td>
<td>• Replaced cpaPmGetInstanceInfo with cpaPmGetInstanceInfo2 that gets a new info structure, CpaInstanceInfo2, which supersedes the previous version. Additional info includes physical ID, core affinity, and NUMA relevant node.</td>
</tr>
<tr>
<td>February 2009</td>
<td>0.74</td>
<td>• Added capabilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add distinction between stateful and stateless. Updated with cpaDcGet/SetMemoryConfiguration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Changed from buffer lists to u32 pointers for responses.</td>
</tr>
<tr>
<td>December 2008</td>
<td>0.73</td>
<td>• Addition of response Arguments - APIs can use source and destination buffers in an easier fashion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Change from flat buffers to buffer lists to align with QA conventions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Major re-write of description of buffer rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Addition of dynamic Huffman trees</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removal of file-based functions. Deemed that this functionality could be built using other buffer-based APIs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clean up of session parameters and various typos</td>
</tr>
</tbody>
</table>
1 Deprecated List

Class _CpaDcSessionSetupData

As of v1.6 of the Compression API, the fileType and deflateWindowSize fields in this structure have been deprecated and should not be used.

Class _CpaInstanceInfo

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceId2.

Global CPA_DEPRECATED

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

Global CPA_DEPRECATED

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

Global CpaDcFileType

As of v1.6 of the Compression API, this enum has been deprecated.

Global CpaDcCompType

As of v1.6 of the Compression API, CPA_DC_LZS, CPA_DC_ELZS and CPA_DC_LZSS have been deprecated and should not be used.

§
2 CPA API

2.1 Detailed Description

File: cpa.h

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

2.2 Modules

- Base Data Types
- CPA Type Definition
- Data Compression API
Figure 2. Collaboration diagram for Base Data Types:

3.1 **Detailed Description**

File: cpa.h

The base data types for the Intel CPA API.

3.2 **Data Structures**

- struct _CpaFlatBuffer
- struct _CpaBufferList
- struct _CpaPhysFlatBuffer
- struct _CpaPhysBufferList
- struct _CpaInstanceId
- struct _CpaInstanceId
- struct _CpaInstanceId
- struct _CpaInstanceId

3.3 **Defines**

- `#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`
3.4 **Typedefs**

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

3.5 **Enumerations**

- `enum _CpaInstanceType {`
3.6 Data Structure Documentation

3.6.1 _CpaFlatBuffer Struct Reference

Detailed Description

Flat buffer structure containing a pointer member and length member.

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

Data Fields

- Cpa32U dataLenInBytes
- Cpa8U * pData

Field Documentation

Cpa32U _CpaFlatBuffer::dataLenInBytes

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

\[ \text{Cpa8U} \_ \text{CpaFlatBuffer::pData} \]

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

### 3.6.2 _CpaBufferList Struct Reference

**Figure 3.** Collaboration diagram for _CpaBufferList:

![Collaboration diagram for _CpaBufferList](image)

**Detailed Description**

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

A scatter/gather buffer list structure. This buffer structure is typically used to represent a region of memory that 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 medata Buffer. The returned size (in bytes) may then be passed in a memory allocation routine to allocate the pPrivateMetaData memory.

**Data Fields**

- Cpa32U numBuffers
- CpaFlatBuffer * pBuffers
Base Data Types

- void * pUserData
- void * pPrivateMetaData

Field Documentation

**Cpa32U_CpaBufferList::numBuffers**
Number of buffers in the list

**CpaFlatBuffer* _CpaBufferList::pBuffers**
Pointer to an unbounded array containing the number of CpaFlatBuffers defined by numBuffers.

**void* _CpaBufferList::pUserData**
This is an opaque field that is not read or modified internally.

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

3.6.3 _CpaPhysFlatBuffer Struct Reference

Detailed Description
Flat buffer structure with physical address.

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

Data Fields
- Cpa32U dataLenInBytes
- Cpa32U reserved
- CpaPhysicalAddr bufferPhysAddr

Field Documentation

**Cpa32U_CpaPhysFlatBuffer::dataLenInBytes**
Data length specified in bytes. When used as an input parameter to a function, the length specifies the current length of the buffer. When used as an output parameter to a function, the length passed in specifies the maximum length of the buffer on return (i.e. the allocated length). The implementation will not write past this length. On return, the length is always unchanged.

**Cpa32U_CpaPhysFlatBuffer::reserved**
Reserved for alignment.

**CpaPhysicalAddr** _CpaPhysFlatBuffer::bufferPhysAddr_

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

### 3.6.4  _CpaPhysBufferList Struct Reference

**Figure 4.** Collaboration diagram for _CpaPhysBufferList_

#### Detailed Description

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

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

#### Data Fields

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

#### Field Documentation

**Cpa64U _CpaPhysBufferList::reserved0**
Reserved for internal usage

\texttt{Cpa32U\_CpaPhysBufferList::numBuffers}

Number of buffers in the list

\texttt{Cpa32U\_CpaPhysBufferList::reserved1}

Reserved for alignment

\texttt{CpaPhysFlatBuffer\_CpaPhysBufferList::flatBuffers[]}

Array of flat buffer structures, of size numBuffers

\section*{3.6.5 \_CpaInstanceInfo Struct Reference}

\textbf{Detailed Description} \textit{Instance Info Structure Deprecated:}

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

Structure that contains the information to describe the instance.

\textbf{Data Fields}

- \texttt{enum\_CpaInstanceType type}
- \texttt{enum\_CpaInstanceState state}
- \texttt{Cpa8U name [CPA\_INSTANCE\_MAX\_NAME\_SIZE\_IN\_BYTES]}
- \texttt{Cpa8U version [CPA\_INSTANCE\_MAX\_VERSION\_SIZE\_IN\_BYTES]}

\textbf{Field Documentation}

\texttt{enum\_CpaInstanceType\_CpaInstanceInfo::type}

Type definition for this instance.

\texttt{enum\_CpaInstanceState\_CpaInstanceInfo::state}

Operational state of the instance.

\texttt{Cpa8U\_CpaInstanceInfo::name[CPA\_INSTANCE\_MAX\_NAME\_SIZE\_IN\_BYTES]}

Simple text string identifier for the instance.

\texttt{Cpa8U\_CpaInstanceInfo::version[CPA\_INSTANCE\_MAX\_VERSION\_SIZE\_IN\_BYTES]}

Version string. There may be multiple versions of the same type of instance accessible through a particular library.
**3.6.6 _CpaPhysicalInstanceId Struct Reference**

**Detailed Description**

Physical Instance ID

Identifies the physical instance of an accelerator execution engine.

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

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

**Data Fields**

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

**Field Documentation**

*Cpa16U__CpaPhysicalInstanceId::packageId*

Identifies the package within which the accelerator is contained.

*Cpa16U__CpaPhysicalInstanceId::acceleratorId*

Identifies the specific accelerator within the package.

*Cpa16U__CpaPhysicalInstanceId::executionEngineId*

Identifies the specific execution engine within the accelerator.

*Cpa16U__CpaPhysicalInstanceId::busAddress*

Identifies the bus address associated with the accelerator execution engine.

*Cpa32U__CpaPhysicalInstanceId::kptAcHandle*

Identifies the achandle of the accelerator.
3.6.7 _CpaInstanceInfo2 Struct Reference

Figure 5. Collaboration diagram for _CpaInstanceInfo2

Detailed Description

Instance Info Structure, version 2

Structure that contains the information to describe the instance.

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]
Base Data Types

- Cpa8U instID [CPA_INST_ID_SIZE]
- CpaPhysicalInstanceId physInstanceId
- Cpa32U nodeAffinity
- CpaOperationalState operState
- CpaBoolean requiresPhysicallyContiguousMemory
- CpaBoolean isPolled
- CpaBoolean isOffloaded

Member Function Documentation

```cpp
_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 hyper-threading (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.

Field Documentation

- CpaAccelerationServiceType _CpaInstanceInfo2::accelerationServiceType
  Type of service provided by this instance.

- Cpa8U _CpaInstanceInfo2::vendorName[CPA_INST_VENDOR_NAME_SIZE]
  String identifying the vendor of the accelerator.

- Cpa8U _CpaInstanceInfo2::partName[CPA_INST_PART_NAME_SIZE]
  String identifying the part (name and/or number).

- Cpa8U _CpaInstanceInfo2::swVersion[CPA_INST_SW_VERSION_SIZE]
String identifying the version of the software associated with the instance. For hardware-based implementations of the API, this should be the driver version. For software-based implementations of the API, this should be the version of the library.

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

```cpp
Cpa8U_CpaInstanceInfo2::instName[CPA_INST_NAME_SIZE]
```

String identifying the name of the instance.

```cpp
Cpa8U_CpaInstanceInfo2::instId[CPA_INST_ID_SIZE]
```

String containing a unique identifier for the instance

```cpp
CpaPhysicalInstanceId_CpaInstanceInfo2::physInstId
```

Identifies the “physical instance” of the accelerator.

```cpp
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.

```cpp
CpaOperationalState_CpaInstanceInfo2::operState
```

Operational state of the instance.

```cpp
CpaBoolean_CpaInstanceInfo2::requiresPhysicallyContiguousMemory
```

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

```cpp
CpaBoolean_CpaInstanceInfo2::isPolled
```

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

```cpp
CpaBoolean_CpaInstanceInfo2::isOffloaded
```

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

### 3.7 Define Documentation

```cpp
#define CPA_INSTANCE_HANDLE_SINGLE
```

Default instantiation handle value where there is only a single instance. Used as an instance handle value where only one instance exists.
#define CPA_DP_BUFLIST

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

#define CPA_STATUS_SUCCESS

Success status value.

#define CPA_STATUS_FAIL

Fail status value.

#define CPA_STATUS_RETRY

Retry status value.

#define CPA_STATUS_RESOURCE

The resource that has been requested is unavailable. Refer to relevant sections of the API for specifics on what the suggested course of action is.

#define CPA_STATUS_INVALID_PARAM

Invalid parameter has been passed in.

#define CPA_STATUS_FATAL

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

#define CPA_STATUS_UNSUPPORTED

The function is not supported, at least not with the specific parameters supplied. This may be because a particular capability is not supported by the current implementation.

#define CPA_STATUS_RESTARTING

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

#define CPA_STATUS_MAX_STR_LENGTH_IN_BYTES

API status string type definition

This type of 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)

#define CPA_STATUS_STR_SUCCESS
Status string for `CPA_STATUS_SUCCESS`.

```c
#define CPA_STATUS_STR_SUCCESS
```

Status string for `CPA_STATUS_FAIL`.

```c
#define CPA_STATUS_STR_FAIL
```

Status string for `CPA_STATUS_RETRY`.

```c
#define CPA_STATUS_STR_RETRY
```

Status string for `CPA_STATUS_RETRY`.

```c
#define CPA_STATUS_STR_RESOURCE
```

Status string for `CPA_STATUS_INVALID_PARAM`.

```c
#define CPA_STATUS_STR_INVALID_PARAM
```

Status string for `CPA_STATUS_FATAL`.

```c
#define CPA_STATUS_STR_FATAL
```

Status string for `CPA_STATUS_UNSUPPORTED`.

```c
#define CPA_STATUS_STR_UNSUPPORTED
```

Maximum instance info name string length in bytes.

```c
#define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES
```

Maximum instance info ID string length in bytes.

```c
#define CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES
```

Maximum instance info version string length in bytes.

```c
#define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES
```

### 3.8 Typedef Documentation

**typedef void* **`CpaInstanceHandle`**

Instance handle type.

Handle used to uniquely identify an instance.

*Note:* Where only a single instantiation exists, this field may be set to `CPA_INSTANCE_HANDLE_SINGLE`.

**typedef Cpa64U CpaPhysicalAddr**

Physical memory address.
Base Data Types

Type for physical memory addresses.

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

Virtual to physical address conversion routine.

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

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

**Assumptions:**
None

**Side-Effects:**
None

**Blocking:**
This function is synchronous and blocking.

**Reentrant:**
No

**Thread-safe:**
Yes

**Parameters:**

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

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

**Postcondition:**
None

**See also:**
None

```c
typedef struct _CpaFlatBuffer CpaFlatBuffer
```

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address. An API instance may require the actual data to be in contiguous physical memory as determined by `CpainstanceInfo2`. 
typedef struct _CpaBufferList CpaBufferList

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

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

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

typedef struct _CpaPhysFlatBuffer CpaPhysFlatBuffer

Flat buffer structure with physical address.

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

typedef struct _CpaPhysBufferList CpaPhysBufferList

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

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

typedef Cpa32S CpaStatus

API status value type definition

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

typedef enum _CpaInstanceType CPA_DEPRECATED

Instance Types

Deprecated:

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

Enumeration of the different instance types.

typedef enum _CpaAccelerationServiceType CpaAccelerationServiceType

Service Type

Enumeration of the different servicetypes.
typedef enum _CpaInstanceState CPA_DEPRECATED

Instance State

Deprecated:

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

Enumeration of the different instance states that are possible.

typedef enum _CpaOperationalState CpaOperationalState

Instance operational state

Enumeration of the different operational states that are possible.

typedef struct _CpaInstanceInfo CPA_DEPRECATED

Instance Info Structure

Deprecated:

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

Structure that contains the information to describe the instance.

typedef struct _CpaPhysicalInstanceId CpaPhysicalInstanceId

Physical Instance ID

Identifies the physical instance of an accelerator execution engine.

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

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

typedef struct _CpaInstanceInfo2 CpaInstanceInfo2

Instance Info Structure, version 2

Structure that contains the information to describe the instance.

typedef enum _CpaInstanceEvent CpaInstanceEvent
Instance Events

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

### 3.9 Enumeration Type Documentation

#### enum _CpaInstanceType

Instance Types

**Deprecated:**

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

Enumeration of the different instance types.

**Enumerator:**

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

#### enum _CpaAccelerationServiceType

Service Type

Enumeration of the different service types.

**Enumerator:**

- CPA_ACC_SVC_TYPE_CRYPTO Cryptography
- CPA_ACC_SVC_TYPE_DATA_COMPRESSION Data Compression
- CPA_ACC_SVC_TYPE_PATTERN_MATCH Pattern Match
- CPA_ACC_SVC_TYPE_RAID RAID
- CPA_ACC_SVC_TYPE_XML XML
- CPA_ACC_SVC_TYPE_VIDEO_ANALYTICS Video Analytics

#### enum _CpaInstanceState

Instance State

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

Enumeration of the different instance states that are possible.

**Enumerator:**

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

```c
enum _CpaOperationalState
{

};
```

Instance operational state

Enumeration of the different operational states that are possible.

**Enumerator:**

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

```c
enum _CpaInstanceState
{

};
```

Instance Events

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

**Enumerator:**

- `CPA_INSTANCE_EVENT_RESTARTING` - Event type that triggers the registered instance notification callback function when an 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 an 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.
4 CPA Type Definition [CPA API]

Figure 6. Collaboration diagram for CPA Type Definition:

| CPA API | CPA Type Definition |

4.1 Detailed Description

File: cpa_types.h

This is the CPA Type Definitions.

4.2 Defines

- #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

4.3 Typedefs

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

- enum _CpaBoolean
  
  {CPA_FALSE,
   CPA_TRUE
  }

4.5 Define Documentation

#define NULL

File: cpa_types.h

NULL definition.

#define CPA_BITMAP ( name,
 sizeInBits )

Declare a bitmap of specified size (in bits).

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

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

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

#define CPA_BITMAP_BIT_TEST ( bitmask,
 bit )

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

#define CPA_BITMAP_BIT_SET ( bitmask,
 bit )

File: cpa_types.h

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

#define CPA_BITMAP_BIT_CLEAR ( bitmask,
 bit )
Clear a specified bit in the specified bitmap. The bitmap may have been declared using CPA_BITMAP.

#define CPA_DEPRECATED

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

### 4.6 Typedef Documentation

typedef uint8_t Cpa8U

File: cpa_types.h

Unsigned byte base type.

typedef int8_t Cpa8S

File: cpa_types.h

Signed byte base type.

typedef uint16_t Cpa16U

File: cpa_types.h

Unsigned double-byte base type.

typedef int16_t Cpa16S

File: cpa_types.h

Signed double-byte base type.

typedef uint32_t Cpa32U

File: cpa_types.h

Unsigned quad-byte base type.

typedef int32_t Cpa32S

File: cpa_types.h

Signed quad-byte base type.

typedef uint64_t Cpa64U

File: cpa_types.h

Unsigned double-quad-byte base type.

typedef int64_t Cpa64S
File: cpa_types.h

Signed double-quad-byte base type.

typedef enum _CpaBoolean CpaBoolean

Boolean type.

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

4.7  Enumeration Type Documentation

enum _CpaBoolean

Boolean type.

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

**Enumerator:**

*CPA_FALSE* False value

*CPA_TRUE* True value

§
5 Data Compression API

5.1 Detailed Description

File: cpa_dc.h

These functions specify the API for Data Compression operations.

Remarks:

5.2 Modules

- Data Compression Batch and Pack API
- Data Compression Chaining API
- Data Compression Data Plane API

5.3 Data Structures

- struct_CpaDcInstanceCapabilities
- struct_CpaDcSessionSetupData
- struct_CpaDcSessionUpdateData
- struct_CpaDcStats
- struct_CpaDcRqResults
- struct_CpaIntegrityCrc
- struct_CpaCrCData
- struct_CpaDcSkipData
- struct_CpaDcOpData
5.4 Defines

- #define CPA_DC_API_VERSION_NUM_MAJOR
- #define CPA_DC_API_VERSION_NUM_MINOR
- #define CPA_DC_CHAIN_CAP_BITMAP_SIZE
- #define CPA_DC_BAD_DATA

5.5 Typedefs

- typedef void * CpaDcSessionHandle
- typedef enum _CpaDcFileType CpaDcFileType
- typedef enum _CpaDcFlush CpaDcFlush
- typedef enum _CpaDcHuffType CpaDcHuffType
- typedef enum _CpaDcCompType CpaDcCompType
- 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 _CpaDcInstanceCapabilities CpaDcInstanceCapabilities
- typedef _CpaDcSessionSetupData CpaDcSessionSetupData
- typedef _CpaDcSessionUpdateData CpaDcSessionUpdateData
- typedef _CpaDcRqResults CpaDcRqResults
- typedef _CpaDcOpData CpaDcOpData
- typedef void(* CpaDcInstanceNotificationCbFunc )(const CpaInstanceHandle instanceHandle, void *pCallbackTag, const CpaInstanceEvent instanceEvent)
5.6 Enumerations

- enum _CpaDcFileType
  
  ```c
  { CPA_DC_FT_ASCII, 
    CPA_DC_FT_CSS, 
    CPA_DC_FT_HTML, 
    CPA_DC_FT_JAVA, 
    CPA_DC_FT_OTHER 
  }
  ```

- enum _CpaDcFlush
  
  ```c
  { CPA_DC_FLUSH_NONE, 
    CPA_DC_FLUSH_FINAL, 
    CPA_DC_FLUSH_SYNC, 
    CPA_DC_FLUSH_FULL 
  }
  ```

- enum _CpaDcHuffType
  
  ```c
  { CPA_DC_HT_STATIC, 
    CPA_DC_HT_PRECOMP, 
    CPA_DC_HT_FULL_DYNAMIC 
  }
  ```

- enum _CpaDcCompType
  
  ```c
  { CPA_DC_LZS, 
    CPA_DC_ELZS, 
    CPA_DC_LZSS, 
    CPA_DC_DEFLATE 
  }
  ```

- enum _CpaDcChecksum
  
  ```c
  { CPA_DC_NONE, 
    CPA_DC_CRC32, 
    CPA_DC_ADLER32, 
    CPA_DC_CRC32_ADLER32 
  }
  ```

- enum _CpaDcSessionDir
  
  ```c
  { CPA_DC_DIR_COMPRESS, 
    CPA_DC_DIR_DECOMPRESS, 
    CPA_DC_DIR_COMPRESS_FLUSH_FINAL, 
    CPA_DC_DIR_DECOMPRESS_FLUSH_FINAL 
  }
  ```
• enum _CpaDcSessionState {
    CPA_DC_STATEFUL,
    CPA_DC_STATELESS
}
• enum _CpaDcCompLvl {
    CPA_DC_L1,
    CPA_DC_L2,
    CPA_DC_L3,
    CPA_DC_L4,
    CPA_DC_L5,
    CPA_DC_L6,
    CPA_DC_L7,
    CPA_DC_L8,
    CPA_DC_L9
}
• enum _CpaDcReqStatus {
    CPA_DC_OK,
    CPA_DC_INVALID_BLOCK_TYPE,
    CPA_DC_BAD_STORED_BLOCK_LEN,
    CPA_DC_TOO_MANY_CODES,
    CPA_DC_INCOMPLETE_CODE_LENS,
    CPA_DC_REPEATLE_LENS,
    CPA_DC_MORE_REPEAT,
    CPA_DC_BAD_LITLEN_CODES,
    CPA_DC_BAD_DIST_CODES,
    CPA_DC_INVALID_CODE,
    CPA_DC_INVALID_DIST,
    CPA_DC_OVERFLOW,
    CPA_DC_SOFTERR,
    CPA_DC_FATALERR,
    CPA_DC_MAX_RESUBITERR,
    CPA_DC_INCOMPLETE_FILE_ERR,
    CPA_DC_WDOG_TIMER_ERR,
    CPA_DC_EP_HARDWARE_ERR,
enum _CpaDcAutoSelectBest { CPA_DC_ASB_DISABLED, CPA_DC_ASB_STATIC_DYNAMIC, CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS, CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS, CPA_DC_ASB_ENABLED }

enum _CpaDcSkipMode { CPA_DC_SKIP_DISABLED, CPA_DC_SKIP_AT_START, CPA_DC_SKIP_AT_END, CPA_DC_SKIP_STRIDE }

5.7 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 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 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)
Data Compression API

- 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 cpaDcGenerateHeader (CpaDcSessionHandle pSessionHandle, CpaFlatBuffer *pDestBuff, Cpa32U *count)
- CpaStatus cpaDcGenerateFooter (CpaDcSessionHandle pSessionHandle, 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)
5.8 Data Structure Documentation

5.8.1 _CpaDcInstanceCapabilities Struct Reference

Detailed Description

Implementation Capabilities Structure

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

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 checksumCRC32
- CpaBoolean checksumAdler32
- CpaBoolean dynamicHuffman
- CpaBoolean dynamicHuffmanBufferReq
- CpaBoolean precompiledHuffman
- CpaBoolean autoSelectBestHuffmanTree
Data Compression API

- Cpa8U validWindowSizeMaskCompression
- Cpa8U validWindowSizeMaskDecompression
- Cpa32U internalHuffmanMem
- CpaBoolean endOfLastBlock
- CpaBoolean reportParityError
- CpaBoolean batchAndPack
- CpaBoolean compressAndVerify
- CpaBoolean compressAndVerifyStrict
- CpaBoolean compressAndVerifyAndRecover
- CpaBoolean integrityCrcs

Member Function Documentation

_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 CpaDcChainCapabilities in cpa_dc_chain.h.

Field Documentation

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSCompression
True if the Instance supports Stateful LZS compression

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSDecompression
True if the Instance supports Stateful LZS decompression.

CpaBoolean _CpaDcInstanceCapabilities::statelessLZSCompression
True if the Instance supports Stateless LZS compression.

CpaBoolean _CpaDcInstanceCapabilities::statelessLZSDecompression
True if the Instance supports Stateless LZS decompression

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSSCompression
True if the Instance supports Stateful LZSS compression

CpaBoolean _CpaDcInstanceCapabilities::statefulLZSSDecompression
True if the Instance supports Stateful LZSS decompression
True if the Instance supports Stateful LZSS decompression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statefulLZSSDecompression}

True if the Instance supports Stateless LZSS compression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statelessLZSSDecompression}

True if the Instance supports Stateless LZSS decompression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statelessELZSDecompression}

True if the Instance supports Stateful Extended LZS compression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statefulELZSDecompression}

True if the Instance supports Stateful Extended LZS decompression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statelessELZSDecompression}

True if the Instance supports Stateful Deflate compression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statefulDeflateDecompression}

True if the Instance supports Stateful Deflate decompression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statelessDeflateDecompression}

True if the Instance supports Stateless Deflate decompression

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::statelessDeflateDecompression}

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

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::checksumCRC32}

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

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::checksumAdler32}

True if the Instance supports dynamic Huffman trees in deflate blocks

\texttt{CpaBoolean\_CpaDcInstanceCapabilities::dynamicHuffmanBufferReq}
True if an Instance specific buffer is required to perform a dynamic Huffman tree deflate request

_CpaBoolean_CpaDcInstanceCapabilities::precompiledHuffman_

True if the Instance supports precompiled Huffman trees in deflate blocks

_CpaBoolean_CpaDcInstanceCapabilities::autoSelectBestHuffmanTree_

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

_Cpa8U_CpaDcInstanceCapabilities::validWindowSizeMaskCompression_

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

_Cpa8U_CpaDcInstanceCapabilities::validWindowSizeMaskDecompression_

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

_Cpa32U_CpaDcInstanceCapabilities::internalHuffmanMem_

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

_CpaBoolean_CpaDcInstanceCapabilities::endOfLastBlock_

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

_CpaBoolean_CpaDcInstanceCapabilities::reportParityError_

True if the instance supports parity error reporting.

_CpaBoolean_CpaDcInstanceCapabilities::batchAndPack_

True if the instance supports 'batch and pack' compression

_CpaBoolean_CpaDcInstanceCapabilities::compressAndVerify_

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

_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.

_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.

`CpaBoolean __CpaDcInstanceCapabilities::integrityCrcs`

True if the instance supports integrity CRC checking in the compression/decompression datapath.

### 5.8.2 __CpaDcSessionSetupData Struct Reference

**Detailed Description**

Session Setup Data.

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

**Deprecated:**

As of v1.6 of the Compression API, the `fileType` and `deflateWindowSize` fields in this structure have been deprecated and should not be used.

**Data Fields**

- `CpaDcCompLvl compLevel`
- `CpaDcCompType compType`
- `CpaDcHuffType huffType`
- `CpaDcAutoSelectBest autoSelectBestHuffmanTree`
- `CpaDcFileType fileType`
- `CpaDcSessionDir sessDirection`
- `CpaDcSessionState sessState`
- `Cpa32U deflateWindowSize`
- `CpaDcChecksum checksum`

**Field Documentation**

- `CpaDcCompLvl __CpaDcSessionSetupData::compLevel`
  Compression Level from CpaDcCompLvl

- `CpaDcCompType __CpaDcSessionSetupData::compType`
  Compression type from CpaDcCompType

- `CpaDcHuffType __CpaDcSessionSetupData::huffType`
  Huffman type from CpaDcHuffType
**CpaDcAutoSelectBest** _CpaDcSessionSetupData::autoSelectBestHuffmanTree_

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

**CpaDcFileType** _CpaDcSessionSetupData::fileType_

File type for the purpose of determining Huffman Codes from CpaDcFileType. As of v1.6 of the Compression API, this field has been deprecated and should not be used.

**CpaDcSessionDir** _CpaDcSessionSetupData::sessDirection_

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

**CpaDcSessionFactory** _CpaDcSessionSetupData::sessState_

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

**Cpa32U** _CpaDcSessionSetupData::deflateWindowSize_

Base 2 logarithm of maximum window size minus 8 (a value of 7 for a 32K window size). Permitted values are 0 to 7. cpaDcDecompressData may return an error if an attempt is made to decompress a stream that has a larger window size. As of v1.6 of the Compression API, this field has been deprecated and should not be used.

**CpaDcChecksum** _CpaDcSessionSetupData::checksum_

Desired checksum required for the session

### 5.8.3 _CpaDcSessionUpdateData Struct Reference

**Detailed Description**

Session Update Data.

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.

**Data Fields**

- CpaDcCompLvl compLevel
- CpaDcHuffType huffType
- CpaBoolean enableDmm

**Field Documentation**

**CpaDcCompLvl** _CpaDcSessionUpdateData::compLevel_

Compression Level from CpaDcCompLvl

**CpaDcHuffType** _CpaDcSessionUpdateData::huffType_

Huffman type from CpaDcHuffType
Data Compression API

CpaBoolean _CpaDcSessionUpdateData::enableDmm
Desired DMM required for the session

5.8.4 _CpaDcStats Struct Reference

Detailed Description
Compression Statistics Data.

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

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

Field Documentation

Cpa64U _CpaDcStats::numCompRequests
Number of successful compression requests

Cpa64U _CpaDcStats::numCompRequestsErrors
Number of compression requests that had errors and could not be processed

Cpa64U _CpaDcStats::numCompCompleted
Compression requests completed

Cpa64U _CpaDcStats::numCompCompletedErrors
Compression requests not completed due to errors

Cpa64U _CpaDcStats::numCompCnvErrorsRecovered
Compression CNV errors that have been recovered

Cpa64U _CpaDcStats::numDecompRequests
Number of successful decompression requests

Cpa64U_CpaDcStats::numDecompRequestsErrors

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

Cpa64U_CpaDcStats::numDecompCompleted

Decompression requests completed

Cpa64U_CpaDcStats::numDecompCompletedErrors

Decompression requests not completed due to errors

5.8.5 _CpaDcRqResults Struct Reference

Detailed Description

Request results data

This structure contains the request results.

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

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

Data Fields

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

Field Documentation

CpaDcReqStatus _CpaDcRqResults::status

Additional status details from accelerator

Cpa32U _CpaDcRqResults::produced

Octets produced by the operation
Cpa32U_CpaDcRqResults::consumed
Octets consumed by the operation

Cpa32U_CpaDcRqResults::checksum
Initial checksum passed into stateless operations. Will also be updated to the checksum produced by the operation

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

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.

5.8.6 _CpaIntegrityCrc Struct Reference

Detailed Description
Integrity CRC calculation details

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

Data Fields
- Cpa32U iCrc
- Cpa32U oCrc

Field Documentation

Cpa32U_CpaIntegrityCrc::iCrc
CRC calculated on request’s input buffer

Cpa32U_CpaIntegrityCrc::oCrc
CRC calculated on request’s output buffer
5.8.7 _CpaCrcData Struct Reference

Figure 8. Collaboration diagram for _CpaCrcData

Detailed Description
Collection of CRC related data

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 'CpIntegrityCrc integrityCrc' field.

Note: This structure must be allocated in physical contiguous memory

Data Fields
- Cpa32U crc32
- Cpa32U adler32
- CpIntegrityCrc integrityCrc

Field Documentation

_Cpa32U _CpaCrcData::crc32
CRC32 calculated on the input buffer during compression requests and on the output buffer during decompression requests.

_Cpa32U _CpaCrcData::adler32
ADLER32 calculated on the input buffer during compression requests and on the output buffer during decompression requests.
CpalntegrityCrc::_CpaCrcData::integrityCrc

Integrity CRCs

5.8.8 _CpaDcSkipData Struct Reference

Detailed Description

Skip Region Data.

This structure contains data relating to configuring skip region behavior. 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.

Data Fields

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

Field Documentation

_CpaDcSkipMode::_CpaDcSkipData::skipMode

Skip mode from CpaDcSkipMode for buffer processing

_Cpa32U::_CpaDcSkipData::skipLength

Number of bytes to skip when skip mode is enabled

_Cpa32U::_CpaDcSkipData::strideLength

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

_Cpa32U::_CpaDcSkipData::firstSkipOffset

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

(De)Compression request input parameters.

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

Data Fields

- CpaDcFlush flushFlag
- CpaBoolean compressAndVerify
- CpaBoolean compressAndVerifyAndRecover
- CpaBoolean integrityCrcCheck
Data Compression API

- CpaBoolean verifyHwIntegrityCrcs
- CpaDcSkipData inputSkipData
- CpaDcSkipData outputSkipData
- CpaCrcData * pCrcData

Field Documentation

**CpaDcFlush** _CpaDcOpData::flushFlag_

Indicates the type of flush to be performed.

**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.

**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.

**CpaBoolean** _CpaDcOpData::integrityCrcCheck_

If set to true, the implementation will verify that data integrity is preserved through the processing pipeline. This behavior supports stateless and stateful behavior for both static and dynamic Huffman encoding.

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

**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.

**CpaDcSkipData** _CpaDcOpData::inputSkipData_

Optional skip regions in the input buffers

**CpaDcSkipData** _CpaDcOpData::outputSkipData_

Optional skip regions in the output buffers
5.9 Define Documentation

```c
#define CPA_DC_API_VERSION_NUM_MAJOR
CPA DC Major Version Number

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

```c
#define CPA_DC_API_VERSION_NUM_MINOR
CPA DC Minor Version Number

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

```c
#define CPA_DC_CHAIN_CAP_BITMAP_SIZE
Size of bitmap needed for compression chaining capabilities.

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).
```

```c
#define CPA_DC_BAD_DATA
Service specific return codes

Compression specific return codes
Input data in invalid
```

5.10 Typedef Documentation

```c
typedef void* CpaDcSessionHandle
Compression API session handle type

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

```c
typedef enum _CpaDcFileType CpaDcFileType
```
Supported file types

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

**Deprecated:**

As of v1.6 of the Compression API, this enum has been deprecated.

```c
typedef enum _CpaDcFlush CpaDcFlush
```

Supported flush flags

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

```c
typedef enum _CpaDcHuffType CpaDcHuffType
```

Supported Huffman Tree types

This enumeration lists support for Huffman Tree types. Selecting Static Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with fixed Huffman trees". Selecting Full Dynamic Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with dynamic Huffman codes". The headers are calculated on the data being compressed, requiring two passes. Selecting Precompiled Huffman Trees generates blocks with RFC 1951 dynamic headers. The headers are precalculated and are specified by the file type.

```c
typedef enum _CpaDcCompType CpaDcCompType
```

Supported compression types

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

**Deprecated:**

As of v1.6 of the Compression API, CPA_DC_LZS, CPA_DC_ELZS and CPA_DC_LZSS have been deprecated and should not be used.

```c
typedef enum _CpaDcChecksum CpaDcChecksum
```

Supported checksum algorithms

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

```c
typedef enum _CpaDcSessionDir CpaDcSessionDir
```

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

```c
typedef enum _CpaDcSessionState CpaDcSessionState
```

Supported session state settings

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

Stateful sessions are limited to have only one in-flight message per session. This means a compression or decompression request must be complete before a new request can be started. This applies equally to sessions that are unidirectional 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.

```c
typedef enum _CpaDcCompLv CpaDcCompLv
```

Supported compression levels

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

```c
typedef enum _CpaDcReqStatus CpaDcReqStatus
```

Supported additional details from accelerator

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

```c
typedef enum _CpaDcAutoSelectBest CpaDcAutoSelectBest
```

Supported modes for automatically selecting the best compression type.

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

```c
typedef enum _CpaDcSkipMode CpaDcSkipMode
```

Supported modes for skipping regions of input or output buffers.

This enumeration lists the supported modes for skipping regions of input or output buffers.
typedef void(* CpaDcCallbackFn)(void *callbackTag, CpaStatus status) Definition of callback function invoked for asynchronous cpaDc requests.

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

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

*callbackTag* User-supplied value to help identify request.

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

**Return values:**

None

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

None

**See also:**

None
### Implementation Capabilities Structure

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

```c
typedef struct _CpaDcInstanceCapabilities CpaDcInstanceCapabilities
```

### Session Setup Data

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

**Deprecated:**

As of v1.6 of the Compression API, the fileType and deflateWindowSize fields in this structure have been deprecated and should not be used.

```c
typedef struct _CpaDcSessionSetupData CpaDcSessionSetupData
```

### Session Update Data

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

```c
typedef struct _CpaDcSessionUpdateData CpaDcSessionUpdateData
```

### Compression Statistics Data

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

```c
typedef struct _CpaDcStats CpaDcStats
```

### Request results data

This structure contains the request results.

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

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

```c
typedef struct _CpaDcRqResults CpaDcRqResults
```

### Integrity CRC calculation details

```c
typedef struct _CpaIntegrityCr CpaIntegrityCr
```
This structure contains information about resulting integrity CRC calculations performed for a single request.

```c
typedef struct _CpaCrcData CpaCrcData
```

Collection of CRC related data

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.

**Note:** This structure must be allocated in physical contiguous memory

```c
typedef struct _CpaDcSkipData CpaDcSkipData
```

Skip Region Data.

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.

```c
typedef struct _CpaDcOpData CpaDcOpData
```

(De)Compression request input parameters.

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

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

Callback function for instance notification support.

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

**Context:**

This function will be executed in a context that requires that sleeping must not be permitted.

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

No

**Reentrant:**

No
Thread-safe:
Yes

Parameters:

[in]  `instanceHandle` Instance handle.

[in]  `pCallbackTag` Opaque value provided by user while making individual function calls.

[in]  `instanceEvent` the event that will trigger this function to get invoked.

Return values:
None

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

Postcondition:
None

Note:
None

See also:
`cpaDcInstanceSetNotificationCb()`.

5.11  Enumeration Type Documentation

```
enum _CpaDcFileType

Supported file types

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

Deprecated:
As of v1.6 of the Compression API, this enum has been deprecated.

Enumerator:

`CPA_DC_FT_ASCII`  ASCII File Type

`CPA_DC_FT_CSS`  Cascading Style Sheet File Type
```
Data Compression API

**CPA_DC_FT_HTML**  HTML or XML (or similar) file type

**CPA_DC_FT_JAVA**  File Java code or similar

**CPA_DC_FT_OTHER**  Other file types

---

**enum_CpaDcFlush**

Supported flush flags

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

**Enumerator:**

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

---

**enum_CpaDcHuffType**

Supported Huffman Tree types

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

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

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

**Enumerator:**

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

---

**enum_CpaDcCompTyp**

Supported compression types

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

**Deprecated:**

As of v1.6 of the Compression API, CPA_DC_LZS, CPA_DC_ELZS and CPA_DC_LZSS have been deprecated and should not be used.

**Enumerator:**

- **CPA_DC_LZS** LZS Compression
- **CPA_DC_ELZS** Extended LZS Compression
- **CPA_DC_LZSS** LZSS Compression
- **CPA_DC_DEFLATE** Deflate Compression

```cpp
enum _CpaDcChecksum

Supported checksum algorithms

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

**Enumerator:**

- **CPA_DC_NONE** No checksums 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

```cpp
enum _CpaDcSessionDir

Supported session directions

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

**Enumerator:**

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

```cpp
enum _CpaDcSessionState

Supported session state settings

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

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

**_CpaDcCompLvl_**

Supported compression levels

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

**Enumerator:**

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

**_CpaDcReqStatus_**

Supported additional details from accelerator

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

**Enumerator:**

- `CPA_DC_OK` No error detected by compression slice
- `CPA_DC_INVALID_BLOCK_TYPE` Invalid block type (type == 3)
- `CPA_DC_BAD_STORED_BLOCK_LEN` Stored block length did not match one's complement
- `CPA_DC_TOO_MANY_CODES` Too many length or distance codes
- `CPA_DC_INCOMPLETE_CODE_LENS` Code length codes incomplete
Data Compression API

**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

**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 block (not supported)

**CPA_DC_CRC_INTEG_ERR**  A data integrity CRC error was detected

```
enum _CpaDcAutoSelectBest
```

Supported modes for automatically selecting the best compression type.

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

**enum **CpaDcSkipMode**

Supported modes for skipping regions of input or output buffers.

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.

### 5.12 Function Documentation

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

Retrieve Instance Capabilities

This function is used to retrieve the capabilities matrix of an instance.
Context:
This function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]  dcInstance    Instance handle derived from discovery functions

[in,out]  pInstanceCapabilities    Pointer to a capabilities struct

Return values:

CPA_STATUS_SUCCESS    Function executed successfully.
CPA_STATUS_FAIL    Function failed.
CPA_STATUS_INVALID_PARAM    Invalid parameter passed in.
CPA_STATUS_RESOURCE    Error related to system resources.
CPA_STATUS_RESTARTING    API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED    Function is not supported.

Precondition:
None

Postcondition:
None

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

See also:
None

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

Initialize compression decompression session

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

Context:

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

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes
**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

**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 bidirectional session is being initialized, then the window size must be valid for both compress and decompress.

See also:
None

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

Compression Session Reset Function.

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.

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`

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

Compression Session Update Function.

This function is used to modify some select compression parameters of a previously initialized session handle. The update would 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.

**Blocking:**

No

**Reentrant:**

No

**Thread-safe:**

Yes
Data Compression API

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_STATUSRESOURCE 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()

CpaStatus cpaDcRemoveSession ( const CpaInstanceHandle dcInstance,
CpaDcSessionHandle pSessionHandle
)

Compression Session Remove Function.

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

Context:

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

Assumptions:
Data Compression API

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()

CpaStatus cpaDcDeflateCompressBound (const CpaInstanceHandle dcInstance,
Deflate Compression Bound API

This function provides the maximum output buffer size for a Deflate compression operation in the "worstcase" (non-compressible) scenario. Its 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.
Data Compression API

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

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

Submit a request to compress a buffer of data.

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

Context:
When called as an asynchronous 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.
Data Compression API

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in] dcInstance Target service instance.

[in, out] pSessionHandle Session handle.


[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 numBufferselement 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 initialized 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 be 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.
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 ensures 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

\texttt{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.

This API consumes data from the input buffer and generates compressed data in the output buffer. This API is very similar to \texttt{cpaDcCompressData()} except it provides a CpaDcOpData structure for passing additional input parameters not covered in \texttt{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.
- **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:
Data Compression API

pSessionHandle has session related state information

**Note:** This function passes control to the compression service for processing

See also:

cpaDcCompressData()

```c
CpaStatus cpaDcDecompressData ( CpainstanceHandle dcinstance,  
CpaDcSessionHandle pSessionHandle,  
CpaBufferList * pSrcBuff,  
CpaBufferList * pDestBuff, CpaDcRqResults * pResults, CpaDcFlush flushFlag,  
void * callbackTag  
)  
```

Submit a request to decompress a buffer of data.

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

**Context:**

When called as an asynchronous 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.
Data Compression API

[in, out] pSessionHandle  Session handle.


[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 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 set up. If a non-NULL value was specified the supplied callback function will be invoked asynchronously with the response of this request, along with the callbackTag specified in the function.

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

See also:

cpaDcCompressData()

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

Submit a request to decompress a buffer of data.

This API consumes compressed data from the input buffer and generates uncompressed data in the output buffer. 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]` `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_STATUSRESOURCE` 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()`

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

Generate compression header.

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

**Context:**

This function may be call from any context.

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

No

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in,out] `pSessionHandle` Session handle.

[in] `pDestBuff` Pointer to data buffer where the compression header will go.

[out] `count` Pointer to counter filled in with header size.

**Return values:**

`CPA_STATUS_SUCCESS` Function executed successfully.

`CPA_STATUS_FAIL` Function failed.
Data Compression API

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

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

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

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

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

**See also:**

None

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

Generate compression footer.

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

**Context:**

This function may be call from any context.

**Assumptions:**

None
Side-Effects:
All session variables are reset

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in, out]  pSessionHandle  Session handle.

[in]  pDestBuff  Pointer to data buffer where the compression footer will go.

[in, out]  pResults  Pointer to results structure filled by CpaDcCompressData. Updated with the results of this API call

Return values:

CPA_STATUS_SUCCESS  Function executed successfully.

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

CPA_STATUS_RESTARTING  API implementation is restarting. Resubmit the request.

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.

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 number’s bytes added to the buffer. The pointer to the buffer will not be modified.

This function is not supported for stateless sessions.

**See also:**
None

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

Retrieve statistics

This API retrieves the current statistics for a compression instance.

**Context:**
This function may be call from any context.

**Assumptions:**
None

**Side-Effects:**
None

**Blocking:**
Yes

**Reentrant:**
No

**Thread-safe:**
Yes

**Parameters:**
[in]  dcInstance  Instance handle.


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

CpaStatus cpaDcGetNumInstances (Cpa16U * pNumInstances)  Get the number of device instances that are supported by the API implementation.

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

Context:

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

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No
Thread-safe:

Yes

Parameters:

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

Return values:

`CPA_STATUS_SUCCESS` Function executed successfully.

`CPA_STATUS_FAIL` Function failed.

`CPA_STATUS_INVALID_PARAM` Invalid parameter passed in.

`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`

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

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

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

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

Context:

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

Assumptions:
Data Compression API

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

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

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

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

Compression Component Initialization and Start function.

This function will initialize and start the compression component. It MUST be called before any
other compress function is called. This function SHOULD be called only once (either for the
very first time, or after a 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:
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.


Return values:

CPA_STATUS_SUCCESS Function executed successfully.

CPA_STATUS_FAIL Function failed. Suggested course of action is to shut down 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()

CpaStatus cpaDcStopInstance ( CpaInstanceHandle instanceHandle )

Compress Component Stop function.

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 shut down.

**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()`

```c
CpaStatus cpaDcInstanceGetInfo2 ( const CpaInstanceHandle instanceHandle,
                                   CpaInstanceInfo2 * pInstanceInfo2
                               )
```
Function to get information on a particular instance.

This function will provide instance specific information through a `CpainstanceInfo2` structure.

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in] `instanceHandle`  Handle to an instance of this API to be initialized.

[out] `pInstanceInfo2`  Pointer to the memory location allocated by the client into which the `CpainstanceInfo2` structure will be written.

**Return values:**

`CPA_STATUS_SUCCESS`  Function executed successfully.

`CPA_STATUS_FAIL`  Function failed.

`CPA_STATUS_INVALID_PARAM`  Invalid parameter passed in.

`CPA_STATUS_UNSUPPORTED`  Function is not supported.

**Precondition:**

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

**Postcondition:**

None
Note: Note

See also:

cpaDcGetNumInstances, cpaDcGetInstances, CpaInstanceInfo2

CpaStatus

cpaDcInstanceSetNotificationCb (const CpaInstanceHandle const

CpaDcInstanceNotificationCb Func instanceHandle, pInstanceNotificationCb,

void * pCallbackTag

)

Subscribe for instance notifications.

Clients of the CpaDc interface can subscribe for instance notifications by registering a

CpaDcInstanceNotificationCb Func 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.
Data Compression API

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

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

CpaStatus cpaDcGetSessionSize (CpaInstanceHandle dcInstance,
);

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 `cpaDataGetSessionSize()` will always return the same session size and that the size will not be different for different setup data parameters. However, it should be noted that the size may change: (1) between different implementations of the API (e.g., between software and hardware implementations or between different hardware implementations) (2) between different releases of the same API implementation.

See also:

`cpaDcInitSession()`

CpaStatus `cpaDataBufferListGetMetaSize (const CpaInstanceHandle instanceHandle, Cpa32U numBuffers, Cpa32U * pSizeInBytes)`

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

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

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

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

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

Context:

This function may be called from any context.
Data Compression API

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
Set Address Translation function

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

**Assumptions:**
None

**Side-Effects:**
None

**Blocking:**
This function is synchronous and blocking.

**Reentrant:**
No

**Thread-safe:**
Yes

**Parameters:**

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

**Return values:**

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

**Precondition:**
None

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

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

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

Context:
No restrictions

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:
[in]  
  dcInstance  Instance handle.

[in]  
  pSessionData  Pointer to a user instantiated structure containing session data.

[out]  
  pSessionSize  On return, this parameter will be the size of the memory that will be required by cpaDcInitSession() for session data.

Return values:
**Data Compression API**

**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 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()`

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

Compression Session Update Function.

This function is used to modify some select compression parameters of a previously initialized session handle for a data plane session. This update would 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()`
- `CpaStatus cpaDcDpRemoveSession (const CpaInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)`
Compression Data Plane Session Remove Function.

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

Context:

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

Assumptions:

None

Side-Effects:

None

Blocking:

No

Reentrant:

No

Thread-safe:

Yes

Parameters:

[in]  dcInstance  Instance handle.

[in,out]  pSessionHandle  Session handle.

Return values:

CPA_STATUS_SUCCESS  Function executed successfully.

CPA_STATUS_FAIL  Function failed.

CPA_STATUS_RETRY  Resubmit the request.

CPA_STATUS_INVALID_PARAM  Invalid parameter passed in.

CPA_STATUS_RESOURCE  Error related to system resources.

CPA_STATUS_RESTARTING  API implementation is restarting. Resubmit the request.

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`
6 Data Compression Batch and Pack API [Data Compression API]

Figure 10. Collaboration diagram for Data Compression Batch and Pack API:

6.1 Detailed Description

File: cpa_dc_bp.h

These functions specify the API for Data Compression operations related to the 'Batch and Pack' mode of operation.

Remarks:

6.2 Data Structures

• struct _CpaDcBatchOpData

6.3 Typedefs

• typedef _CpaDcBatchOpData CpaDcBatchOpData

6.4 Functions

• CpaStatus cpaDcBPCompressData (CpainstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, const Cpa32U numRequests, CpaDcBatchOpData *pBatchOpData, CpaBufferList *pDestBuff, CpaDcRqResults *pResults, void *callbackTag)

• CpaStatus cpaDcBnpBufferListGetMetaSize (const CpainstanceHandle instanceHandle, Cpa32U numJobs, Cpa32U *pSizeInBytes)
6.5 Data Structure Documentation

6.5.1 _CpaDcBatchOpData Struct Reference

Figure 11. Collaboration diagram for _CpaDcBatchOpData

6.5.1.1 Detailed Description

Batch request input parameters.

This structure contains the request information for use with batched compression operations.
6.5.2 Data Fields
- CpaDcOpData opData
- CpaBufferList * pSrcBuff
- CpaBoolean resetSessionState

6.5.3 Field Documentation

CpaDcOpData _CpaDcBatchOpData::opData
Compression input parameters

CpaBufferList* _CpaDcBatchOpData::pSrcBuff
Input buffer list containing the data to be compressed.

CpaBoolean _CpaDcBatchOpData::resetSessionState
Reset the session state at the beginning of this request within the batch. Only applies to stateful sessions. When this flag is set, the history from previous requests in this session will not be used when compressing the input data for this request in the batch.

6.6 Typedef Documentation

typedef struct _CpaDcBatchOpData CpaDcBatchOpData
Batch request input parameters.
This structure contains the request information for use with batched compression operations.

6.7 Function Documentation

CpaStatus cpaDcBPCompressData ( CpaInstanceHandle dcInstance,
CpaDcSessionHandle pSessionHandle, const Cpa32U numRequests,
CpaDcBatchOpData * pBatchOpData, CpaBufferList * pDestBuff, CpaDcRqResults * pResults,
void * callbackTag
)
Submit a batch of requests to compress a batch of input buffers into a common output buffer. The same output buffer is used for each request in the batch. This is termed ‘batch and pack’.

This API consumes data from the input buffer and generates compressed data in the output buffer. This API compresses a batch of input buffers and concatenates the compressed data into the output buffer. A results structure is also generated for each request in the batch.
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] numRequests Number of requests in the batch.
[in] pBatchOpData Pointer to an array of CpaDcBatchOpData structures which contain the input buffers and parameters for each request in the batch. There should be numRequests entries in the array.
[in] pDestBuff Pointer to buffer space for data after compression.
[in,out] pResults Pointer to an array of results structures. There should be numRequests entries in the array.
[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.
Data Compression Batch and Pack API [Data Compression API]

**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()` Session must be setup as a stateless session.

**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 buffers in the input list has not been compressed. This situation will occur when there is insufficient space in the output buffer. The calling application should note the number of buffers processed, and then submit the request again, with a new output buffer and with the input buffer list containing the buffers that were 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 number of buffers 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 buffers consumed from the input buffer list.

Destination Buffer Setup Rules
1. The destination buffer list must have storage for processed data and for the packed header information. This means that at least two flat buffers must exist in the buffer list. The first buffer entry will be used for the header information. Subsequent entries will be used for the compressed data.

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.

2. On return, the pResults->produced will indicate the number of bytes of compressed data written to the output buffers. Note that this will not include the header information buffer.

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.

Synchronous or Asynchronous operation of the API is determined by the value of the callbackFn parameter passed to cpaDcInitSession() when the sessionHandle was set up. 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, 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 asynchronously, the return code represents the success or not of asynchronously scheduling the request. The results of the operation, along with the amount of data consumed and produced become available when the callback function is invoked. As such, pResults->consumed and pResults->produced are available only when the operation is complete.

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

See also:
None

CpaStatus cpaDcBnpBufferListGetMetaSize (const CpaInstanceHandle instanceHandle,
Function to return the size of the memory which must be allocated for the pPrivateMetaData member of CpaBufferList contained within CpaDcBatchOpData.

This function is used to obtain the size (in bytes) required to allocate a buffer descriptor for the pPrivateMetaData member in the CpaBufferList structure when Batch and Pack API are used. Should the function return zero then no metadata 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]** `numJobs` the number of jobs defined in the CpaDcBatchOpData table.
- **[out]** `pSizeInBytes` Pointer to the size in bytes of memory to be allocated when the client wishes to allocate a cpaFlatBuffer and the Batch and Pack OP data.

**Return values:**

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

**Precondition:**
None

**Postcondition:**

None

**Note:** None

See also:

`cpaDcBPCompressData()`
7 Data Compression Chaining API

7.1 Detailed Description

File: cpa_dc_chain.h

These functions specify the API for Data Compression Chaining operations.

Remarks:

7.2 Data Structures

- struct _CpaDcChainSessionSetupData
- struct _CpaDcChainOpData
- struct _CpaDcChainRqResults

7.3 Typedefs

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

7.4 Enumerations

- enum _CpaDcChainOperations { CPA_DC_CHAIN_COMPRESS_THEN_HASH, CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT, CPA_DC_CHAIN_COMPRESS_THEN_HASH_ENCRYPT, CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT_HASH, CPA_DC_CHAIN_COMPRESS_THEN_AEAD, CPA_DC_CHAIN_HASH_THEN_COMPRESS, }
7.5 Functions

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

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

- `CpaStatus cpaDcChainResetSession (const CpalInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)`

- `CpaStatus cpaDcChainRemoveSession (const CpalInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle)`

- `CpaStatus cpaDcChainPerformOp (CpalInstanceHandle dcInstance, CpaDcSessionHandle pSessionHandle, CpaBufferList *pSrcBuff, CpaBufferList *pDestBuff, CpaDcChainOperations operation, Cpa8U numOpDatas, CpaDcChainOpData *pChainOpData, CpaDcChainRqResults *pResults, void *callbackTag)`
7.6 Data Structure Documentation

7.6.1 _CpaDcChainSessionSetupData Struct Reference

**Figure 13.** Collaboration diagram for _CpaDcChainSessionSetupData

#### 7.6.1.1 Detailed Description

Chaining Session Setup Data.

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.

#### 7.6.1.2 Data Fields

- CpaDcChainSessionType sessType
- CpaDcSessionSetupData * pDcSetupData
- CpaCySymSessionSetupData * pCySetupData

#### 7.6.1.3 Field Documentation

- **CpaDcSessionSetupData* _CpaDcChainSessionSetupData::pDcSetupData**

  Pointer to compression session setup data

- **CpaCySymSessionSetupData* _CpaDcChainSessionSetupData::pCySetupData**
Pointer to symmetric crypto session setup data

7.6.2 _CpaDcChainOpData Struct Reference

Figure 14. Collaboration diagram for _CpaDcChainOpData:
7.6.2.1 Detailed Description

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

7.6.2.2 Data Fields

- CpaDcChainSessionType opType
- CpaDcOpData * pDcOp
- CpaCySymOpData * pCySymOp

7.6.2.3 Field Documentation

CpaDcChainSessionType _CpaDcChainOpData::opType
Indicate the type for this operation

CpaDcOpData* _CpaDcChainOpData::pDcOp
Pointer to compression operation data

CpaCySymOpData* _CpaDcChainOpData::pCySymOp
Pointer to symmetric crypto operation data

7.6.3 _CpaDcChainRqResults Struct Reference

7.6.3.1 Detailed Description

Chaining request results data
This structure contains the request results.

7.6.3.2 Data Fields

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

**CpaDcReqStatus**::dcStatus
Additional status details from compression accelerator

**CpaStatus**::cyStatus
Additional status details from symmetric crypto accelerator

**CpaBoolean**::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

**Cpa32U**::produced
Octets produced to the output buffer

**Cpa32U**::consumed
Octets consumed from the input buffer

**Cpa32U**::crc32
crc32 checksum produced by chaining operations

**Cpa32U**::adler32
adler32 checksum produced by chaining operations

7.7 Typedef Documentation

**typedef enum** CpaDcChainOperations
Supported operations for compression chaining

This enumeration lists the supported operations for compression chaining

**typedef enum** CpaDcChainSessionType
Supported session types for data compression chaining.

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

**typedef struct** CpaDcChainSessionSetupData
Chaining Session Setup Data.

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.
typedef struct _CpaDcChainOpData CpaDcChainOpData

Compression chaining request input parameters.

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

typedef struct _CpaDcChainRqResults CpaDcChainRqResults

Chaining request results data

This structure contains the request results.

7.8 Enumeration Type Documentation

enum_CpaDcChainOperations

Supported operations for compression chaining

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: first entry is for compression setup data, second 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: first entry is for compression setup data, second entry is for encryption setup data

**CPA_DC_CHAIN_COMPRESS_THEN_HASH_ENCRYPT** 2 operations for chaining: first operation is to perform compression on plain text, second operation is to perform hash on compressed text and encryption on compressed text < 2 entries in CpaDcChainSessionSetupData array: first entry is for compression setup data, second entry is for hash and encryption setup data

**CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT_HASH** 2 operations for chaining: first operation is to perform compression on plain text, second operation is to perform encryption on compressed text and hash on compressed & encrypted text < 2 entries in CpaDcChainSessionSetupData array: first entry is for compression setup data, second 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: first entry is for compression setup data, second 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: first entry is for hash set up data, second entry is for compression setup data

CPA_DC_CHAIN_HASH_VERIFY_THEN_DECOMPRESS 2 operations for chaining: first operation is to perform hash verify on compressed text 2nd operation is to perform decompression on compressed text < 2 entries in CpaDcChainSessionSetupData array: first entry is for hash setup data 2nd entry is for decompression setup data

CPA_DC_CHAIN_DECRYPT_THEN_DECOMPRESS 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

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 perform 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

enum _CpaDcChainSessionType

Supported session types for data compression chaining.

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
7.9 Function Documentation

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.

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

Initialize data compression chaining session

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 is guaranteed to be in the same order the compression requests 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:

[\textbf{[in]} \ dcInstance] Instance handle derived from discovery functions.

[\textbf{[in, out]} \ pSessionHandle] Pointer to a session handle.

[\textbf{[in]} \ operation] the operations for chaining

[\textbf{[in]} \ numSessions] Number of sessions for chaining

[\textbf{[in, out]} \ pSessionData] Pointer to an array of CpaDcChainSessionSetupData structures. There should be numSessions entries in the array.

[\textbf{[in]} \ callbackFn] for synchronous operation this callback shall be a null pointer.

Return values:

\textbf{CPA\_STATUS\_SUCCESS} Function executed successfully.

\textbf{CPA\_STATUS\_FAIL} Function failed.

\textbf{CPA\_STATUS\_INVALID\_PARAM} Invalid parameter passed in.

\textbf{CPA\_STATUS\_RESOURCE} Error related to system resources.

\textbf{CPA\_STATUS\_RESTARTING} API implementation is restarting. Resubmit the request.

\textbf{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 asymmetric crypto session set up data.

2. The supported chaining operations are listed in CpaDcChainOperations. This enum indicates thenumber 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_DCCOMPRESS_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_DCCOMPRESS 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_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

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

Reset a compression chaining session.

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()`

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

Remove a compression chaining session.

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()
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.

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.
[out] pDestBuff Pointer to output data buffer.
[in]  operation Operation for the chaining request
[in] numOpDatas The entries size CpaDcChainOpData 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_DCCHAIN_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.

6. Stateful compression is not supported for chaining.

7. 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:

8. pResults->crc32 is available to application if CpaDcSessionSetupData->checksum is set to CPA_DC_CRC32

9. pResults->adler32 is available to application if CpaDcSessionSetupData->checksum is set to CPA_DC_ADLER32

10. 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
8 Data Compression Data Plane API

Figure 15. Collaboration diagram for Data Compression Data Plane API:

![Collaboration diagram for Data Compression Data Plane API](image)

8.1 Detailed Description

File: cpa_dc_dp.h

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

This API is recommended for data plane applications, in which the cost of offload - that is, the cycles consumed by the driver in sending requests to the hardware, and processing responses - needs to be minimized. 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).
8.2 Data Structures

- struct _CpaDcDpOpData

8.3 Typedefs

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

8.4 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 cpaDcDpPerformOpNow (CpainstanceHandle dcInstance)
8.5 Data Structure Documentation

8.5.1 _CpaDcDpOpData Struct Reference

Figure 16. Collaboration diagram for _CpaDcDpOpData

8.5.1.1 Detailed Description

Operation Data for compression data plane API.

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

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
8.5.1.2 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

8.5.1.3 Field Documentation

_CpaDcDpOpData::reserved0

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

_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).

_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).

_CpaDcDpOpData::reserved1

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

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

Results of the operation. Contents are valid upon completion.

Instance to which the request is to be enqueued

DC Session associated with the stream of requests

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.

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.

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.

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.

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

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.

**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 iset to CPA_TRUE.

**CpaStatus _CpaDcDpOpData::responseStatus**

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

**CpaPhysicalAddr _CpaDcDpOpData::thisPhys**

Physical address of this datastructure

**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 thereresponse later.

### 8.6 Typedef Documentation

**typedef struct _CpaDcDpOpData CpaDcDpOpData**

Operation Data for compression data plane API.

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

The physical memory to which this structure points should be at least 8-byte aligned. All reserved fields SHOULD NOT be written or read by the calling code.

**See also:**

*cpaDcDpEnqueueOp, cpaDcDpEnqueueOpBatch*

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

Definition of callback function for compression data plane API.

This is the callback function prototype. The callback function is registered by the application using the *cpaDcDpRegCbFunc* function call and called back on completion of asynchronous requests made via callsto *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 that 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

8.7 Function Documentation

CpaStatus cpaDcDpInitSession ( CpaInstanceHandle dcInstance,
CpaDcSessionHandle pSessionHandle,
CpaDcSessionSetupData *pSessionData )
Initialize compression or decompression data plane session.

This function is used to initialize a compression/decompression session. A single session can be used for both compression and decompression requests. Clients MUST register a callback function for the compression service using this function. This function returns a unique session handle each time this function is invoked. The order of the callbacks is guaranteed to be in the same order, the compression requests 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 bidirectional 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

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

Registration of the operation completion callback function.

This function allows a completion callback function to be registered. The registered callback function is invoked on completion of 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`

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

Enqueue a single compression or decompression request.

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

The following restrictions apply to the `pOpData` parameter:

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

**Context:**

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

**Side-Effects:**

None

**Blocking:**

No

**Reentrant:**

No

**Thread-safe:**

No

**Parameters:**

- **[in]** `pOpData` Pointer to a structure containing the request parameters. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback, which was registered on the instance via `cpaDcDpRegCbFunc`. See the previous 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`. 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`

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

Enqueue multiple requests to the compression data plane API.

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

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

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

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

**Context:**

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

**Assumptions:**
Client MUST allocate the request parameters to 8 byte alignment. Reserved elements of the CpaDcDpOpData structure MUST not use. 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 earlier 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*. The instance identified by *pOpData[i]->dcInstance* has had a callback function registered via *cpaDcDpRegCbFunc*.
Postcondition:
None

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

See also:
cpaDcDpEnqueueOp

CpaStatus cpaDcDpPerformOpNow (CpainstanceHandle dcInstance)
Submit any previously enqueued requests to be performed now on the compression data plane API.
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`