



Intel® EP80579 Software on Intel® QuickAssist Technology Debug Services API Reference

Automatically generated from sources, July 28, 2008.

Reference Number: 320185, Revision -001

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

This document contains information on products in the design phase of development.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. ("products") in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as "commercial" names for products. Also, they are not intended to function as trademarks.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel Leap ahead., Intel Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

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

Copyright © Intel Corporation 2008. All Rights Reserved.

Revision History

Date	Revision	Description
July 08 2008	-001	Initial Released Version

Table of Contents

1 Acceleration API	1
1.1 Detailed Description.....	1
1.2 Modules.....	1
2 Base Data Types [Acceleration API]	2
2.1 Detailed Description.....	2
2.2 Data Structures.....	2
2.3 Defines.....	2
2.4 Typedefs.....	3
2.5 Enumerations.....	3
2.6 Data Structure Documentation.....	3
2.6.1 icp_flat_buffer_s Struct Reference.....	3
2.7 Define Documentation.....	4
2.8 Typedef Documentation.....	6
2.9 Enumeration Type Documentation.....	8
3 Debug Common Component API	9
3.1 Detailed Description.....	9
3.2 Data Structures.....	9
3.3 Typedefs.....	9
3.4 Functions.....	9
3.5 Data Structure Documentation.....	10
3.5.1 icp_dcc_liveness_status_s Struct Reference.....	10
3.5.2 icp_dcc_package_ver_s Struct Reference.....	11
3.5.3 icp_dcc_module_ver_s Struct Reference.....	11
3.5.4 icp_dcc_version_buffer_s Struct Reference.....	12
3.6 Typedef Documentation.....	13
3.7 Function Documentation.....	14
4 Debug Base Data Structures	23
4.1 Detailed Description.....	23
4.2 Data Structures.....	23
4.3 Defines.....	23
4.4 Typedefs.....	23
4.5 Enumerations.....	23
4.6 Data Structure Documentation.....	24
4.6.1 icp_dcc_thread_id_s Struct Reference.....	24
4.6.2 icp_dcc_ver_info_s Struct Reference.....	24
4.6.3 icp_dcc_sen_msg_s Struct Reference.....	25
4.7 Define Documentation.....	26
4.8 Typedef Documentation.....	27
4.9 Enumeration Type Documentation.....	28

1 Acceleration API

Collaboration diagram for Acceleration API:



1.1 Detailed Description

This is the top level API definition.

It contains structures, data types and definitions that are common across the other interfaces.

1.2 Modules

- **Base Data Types**

The base data types for the Intel Acceleration API.

2 Base Data Types

[Acceleration API]

Collaboration diagram for Base Data Types:



2.1 Detailed Description

The base data types for the Intel Acceleration API.

2.2 Data Structures

- struct **icp_flat_buffer_s**
Flat buffer structure containing a pointer and length member.

2.3 Defines

- #define **NULL**
NULL definition.
- #define **TRUE**
True value definition.
- #define **FALSE**
False value definition.
- #define **ICP_INVALID_HANDLE**
Invalid handle.
- #define **ICP_ACCEL_HANDLE_DEFAULT**
Default acceleration handle value.
- #define **ICP_INVALID_BUFFER_HANDLE**
Invalid buffer handle.
- #define **ICP_STATUS_SUCCESS**
Success status value.
- #define **ICP_STATUS_FAIL**
Fail status value.
- #define **ICP_STATUS_RETRY**
Retry status value.
- #define **ICP_STATUS_RESOURCE**
The resource that has been requested is unavailable status value.
- #define **ICP_STATUS_INVALID_PARAM**
Invalid parameter has been passed in status value.
- #define **ICP_STATUS_FATAL**
A serious error has occurred status value.
- #define **ICP_STATUS_UNDERFLOW**
Underflow error status value - the client is under submitting data.
- #define **ICP_STATUS_OVERFLOW**
Overflow error status value - the client is over submitting data.
- #define **ICP_STATUS_NULL_PARAM**
One or more parameters is null status value.
- #define **ICP_STATUS_MUTEX**
Failure with a mutex operation status value.

2.3 Defines

- **#define ICP_STATUS_ALREADY_REGISTERED**
An attempt was made to register an item, for example a callback, with the same value as an existing registered value.
- **#define ICP_STATUS_INVALID_HANDLE**
An invalid handle was passed in status value.
- **#define ICP_STATUS_NOT_SUPPORTED**
Operation not supported in the current implementation status value.
- **#define ICP_ITERATOR_FIRST**
Use this define to access the first element of a table or other data structure.
- **#define ICP_ITERATOR_LAST**
Use this define to indicate the last iteration of a table or other data structure.

2.4 Typedefs

- **typedef void * icp_user_context_t**
User provided representation of the context that is returned unmodified.
- **typedef void * icp_correlator_t**
User provided correlator value that is returned unmodified.
- **typedef uint32_t icp_handle_t**
Generic handle to items in the Acceleration API.
- **typedef void * icp_accel_handle_t**
Accelerator handle type.
- **typedef icp_handle_t icp_callback_handle_t**
Handle to registered callback function.
- **typedef icp_handle_t icp_event_func_handle_t**
Handle to registered event handler function.
- **typedef uint64_t icp_buffer_handle_t**
Buffer Handle.
- **typedef icp_flat_buffer_s icp_flat_buffer_t**
Flat buffer structure containing a pointer and length member.
- **typedef uint32_t icp_status_t**
Acceleration API status value type definition.
- **typedef uint32_t icp_iterator_t**
Iterator type.

2.5 Enumerations

- **enum icp_boolean_t {**
 ICP_FALSE,
 ICP_TRUE
 }
 Boolean type.

2.6 Data Structure Documentation

2.6.1 icp_flat_buffer_s Struct Reference

2.6.1.1 Detailed Description

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address however the actual data pointed to is required to be in contiguous physical memory. It is expected that this buffer handle will be used when simple,

2.6.1 icp_flat_buffer_s Struct Reference

unchained buffers are needed. The `icp_buffer_handle_t` defined in `icp_buffer.h` describe much more fully featured buffers that may be used when chaining or mapping from OS specific buffers is required.

2.6.1.2 Data Fields

- `uint8_t * pData`
The data pointer is a virtual address, however the actual data pointed to is required to be in contiguous physical memory.
- `uint32_t dataLenInBytes`
Data length specified in bytes.
- `void * clientBufferHandle`
This is an opaque field that is not read or modified internally.

2.6.1.3 Field Documentation

`uint8_t* icp_flat_buffer_s::pData`

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

`uint32_t icp_flat_buffer_s::dataLenInBytes`

Data length specified in bytes.

`void* icp_flat_buffer_s::clientBufferHandle`

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

An example usage scenario for this structure member is for the client to assign this to be the pointer for the start of the buffer in which the data resides. Subsequently it may be used to recover the start of the data buffer.

2.7 Define Documentation

`#define NULL`

NULL definition.

This define is used to identify a NULL value.

`#define TRUE`

True value definition.

`#define FALSE`

False value definition.

`#define ICP_INVALID_HANDLE`

Invalid handle.

This define is used to identify a handle as invalid.

Note:

To mark a buffer handle as invalid use the **ICP_INVALID_BUFFER_HANDLE**.

2.7 Define Documentation

```
#define ICP_ACCEL_HANDLE_DEFAULT
```

Default acceleration handle value.

Used as an acceleration handle value where only one acceleration device exists on silicon.

```
#define ICP_INVALID_BUFFER_HANDLE
```

Invalid buffer handle.

This define is used to identify a buffer handle as invalid.

```
#define ICP_STATUS_SUCCESS
```

Success status value.

```
#define ICP_STATUS_FAIL
```

Fail status value.

```
#define ICP_STATUS_RETRY
```

Retry status value.

```
#define ICP_STATUS_RESOURCE
```

The resource that has been requested is unavailable status value.

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

```
#define ICP_STATUS_INVALID_PARAM
```

Invalid parameter has been passed in status value.

```
#define ICP_STATUS_FATAL
```

A serious error has occurred status value.

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

```
#define ICP_STATUS_UNDERFLOW
```

Underflow error status value - the client is under submitting data.

This status value will be deprecated in a subsequent release of this interface.

```
#define ICP_STATUS_OVERFLOW
```

Overflow error status value - the client is over submitting data.

This status value will be deprecated in a subsequent release of this interface.

```
#define ICP_STATUS_NULL_PARAM
```

One or more parameters is null status value.

This status value will be deprecated in a subsequent release of this interface.

```
#define ICP_STATUS_MUTEX
```

Failure with a mutex operation status value.

This status value will be deprecated in a subsequent release of this interface.

2.8 Typedef Documentation

```
#define ICP_STATUS_ALREADY_REGISTERED
```

An attempt was made to register an item, for example a callback, with the same value as an existing registered value.

This status value will be deprecated in a subsequent release of this interface.

```
#define ICP_STATUS_INVALID_HANDLE
```

An invalid handle was passed in status value.

This status value will be deprecated in a subsequent release of this interface.

```
#define ICP_STATUS_NOT_SUPPORTED
```

Operation not supported in the current implementation status value.

This status value will be deprecated in a subsequent release of this interface.

```
#define ICP_ITERATOR_FIRST
```

Use this define to access the first element of a table or other data structure.

```
#define ICP_ITERATOR_LAST
```

Use this define to indicate the last iteration of a table or other data structure.

2.8 Typedef Documentation

```
typedef void* icp_user_context_t
```

User provided representation of the context that is returned unmodified.

This type defines an opaque user context passed in by the user when the callback handler is registered. It is returned to the user when the callback function is invoked. This value is not modified or used by the acceleration components. It may be used to allow the application to associate a completion callback call with a specific instance of the original function call.

See also:

```
typedef void* icp_correlator_t
```

User provided correlator value that is returned unmodified.

This type defines an opaque value provided by the user while making an API function call. The value is returned to the user when the callback function is invoked. This value is not modified or used internally by the components. It may be used by the client to help correlate a particular instance of a function call to a particular callback.

```
typedef uint32_t icp_handle_t
```

Generic handle to items in the Acceleration API.

This type is a handle that uniquely identifies items in the acceleration API.

Note:

To mark a handle as invalid use **ICP_INVALID_HANDLE**.

```
typedef void* icp_accel_handle_t
```

2.8 Typedef Documentation

Accelerator handle type.

Handle used to uniquely identify an acceleration device instance

Note:

Where only a single accelerator exists on the Silicon variant, this field must be set to **ICP_ACCEL_HANDLE_DEFAULT**.

```
typedef icp_handle_t icp_callback_handle_t
```

Handle to registered callback function.

This type is a handle that uniquely identifies a registered callback function.

```
typedef icp_handle_t icp_event_func_handle_t
```

Handle to registered event handler function.

This type is a handle that uniquely identifies a registered event handler function

```
typedef uint64_t icp_buffer_handle_t
```

Buffer Handle.

This type uniquely identifies a buffer handle for this API.

Purpose:

The intention is to present an abstraction that hides from the clients of the API any of the private, implementation-specific aspects of the buffer format used by implementations of this API. Functions will be provided to translate between this buffer format and those buffer formats used by clients (e.g. OS-specific buffer formats such as sk_bufs, mbufs, etc.), as well as functions to get key data about the buffer, i.e. the length, pointer to the data contained in the buffer, etc. The abstraction also supports the concept of buffer chaining.

Note:

The buffer translation module can be used to perform conversions between various OS buffer formats and the `icp_buffer_handle_t`. Please refer to the buffer translation module documentation for more information on supported data buffer conversions. To define an invalid buffer handle use the **ICP_INVALID_BUFFER_HANDLE** define.

```
typedef struct icp_flat_buffer_s icp_flat_buffer_t
```

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, `pData`, is a virtual address however the actual data pointed to is required to be in contiguous physical memory. It is expected that this buffer handle will be used when simple, unchained buffers are needed. The `icp_buffer_handle_t` defined in `icp_buffer.h` describe much more fully featured buffers that may be used when chaining or mapping from OS specific buffers is required.

```
typedef uint32_t icp_status_t
```

Acceleration API status value type definition.

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

```
typedef uint32_t icp_iterator_t
```

Iterator type.

2.9 Enumeration Type Documentation

This type is used in this API, when the same function is repeatedly called to get values from a table or other data structure.

In the first call, the iterator is set to `ICP_ITERATOR_FIRST`. The API returns an updated value for the iterator which must be passed into the next call until the API returns an iterator value of `ICP_ITERATOR_LAST`

2.9 Enumeration Type Documentation

enum **icp_boolean_t**

Boolean type.

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

Enumerator:

ICP_FALSE False value.

ICP_TRUE True value.

3 Debug Common Component API

3.1 Detailed Description

This section describes the data structures, typedefs, and functions used by the Debug clients.

3.2 Data Structures

- struct **icp_dcc_liveness_status_s**
Thread response Status .
- struct **icp_dcc_package_ver_s**
Structure of Package version in buffer returned to Debug Client.
- struct **icp_dcc_module_ver_s**
Structure of Module version in buffer returned to Debug Client.
- struct **icp_dcc_version_buffer_s**
Structure of Version buffer returned to Debug Client.

3.3 Typedefs

- typedef **icp_dcc_liveness_status_s icp_dcc_liveness_status_t**
Thread response Status .
- typedef **icp_dcc_package_ver_s icp_dcc_package_ver_t**
Structure of Package version in buffer returned to Debug Client.
- typedef **icp_dcc_module_ver_s icp_dcc_module_ver_t**
Structure of Module version in buffer returned to Debug Client.
- typedef **icp_dcc_version_buffer_s icp_dcc_version_buffer_t**
Structure of Version buffer returned to Debug Client.
- typedef void(* **icp_DccSenHandler**)(**icp_dcc_sen_msg_t** *pSenMsg)
Prototype of SEN event callback handler to be registered by user.

3.4 Functions

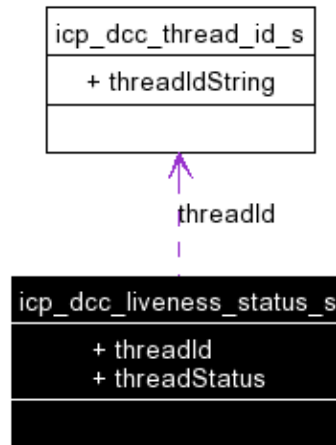
- **icp_status_t icp_DccVersionInfoSizeGet** (uint32_t *pVersionBufferSize)
Get the maximum buffer size needed to retrieve Version information of the Package and all the Components registered with the DCC.
- **icp_status_t icp_DccSoftwareVersionGet** (uint8_t *pVerInfo, uint32_t *pVersionBufferSize)
Returns Package and Components version information.
- **icp_status_t icp_DccLivenessConfigureTimeout** (uint32_t livenessTimeOut)
Set the response timeout value for DCC.
- **icp_status_t icp_DccLivenessResponseSizeGet** (uint32_t *pLivenessResponseSize)
Get the buffer size to be allocated by the DCC user for getting the system response information.
- **icp_status_t icp_DccLivenessVerify** (uint8_t *pLivenessStatus, uint32_t *pBufferSize)
Verify System response.
- **icp_status_t icp_DccDataDumpInfoGet** (uint32_t *noOfModules, uint32_t *pDataDumpSize)
Get the number of Modules and the maximum buffer size required to get a data dump.
- **icp_status_t icp_DccDataDumpGet** (uint32_t moduleId, uint32_t *pDataDumpSize, uint8_t *pDataDump)
Get the number of Modules and the maximum buffer size required to get a data dump.
- **icp_status_t icp_DccSenHandlerRegister** (**icp_DccSenHandler** senHandler)
Register the SEN event callback handler.
- **icp_status_t icp_DccSenHandlerUnregister** (void)

Unregister the SEN event callback handler.

3.5 Data Structure Documentation

3.5.1 icp_dcc_liveness_status_s Struct Reference

Collaboration diagram for icp_dcc_liveness_status_s:



3.5.1.1 Detailed Description

Thread response Status .

This value denotes the response status of the specified thread ID.

Purpose:

During response check, the response status of each thread is returned through this structure.

3.5.1.2 Data Fields

- **icp_dcc_thread_id_t threadId**
Thread ID registered.
- **icp_dcc_thread_status_t threadStatus**
ICP_DCC_THREAD_ID_LIVE or ICP_DCC_THREAD_ID_DEAD.

3.5.1.3 Field Documentation

icp_dcc_thread_id_t icp_dcc_liveness_status_s::threadId

Thread ID registered.

icp_dcc_thread_status_t icp_dcc_liveness_status_s::threadStatus

ICP_DCC_THREAD_ID_LIVE or ICP_DCC_THREAD_ID_DEAD.

3.5.2 icp_dcc_package_ver_s Struct Reference

3.5.2 icp_dcc_package_ver_s Struct Reference

3.5.2.1 Detailed Description

Structure of Package version in buffer returned to Debug Client.

This structure provides Package version information in buffer provided by the Debug Client.

Purpose:

When queried for software version, the version information of the Package is returned in this format.

3.5.2.2 Data Fields

- `uint8_t packageSize`
Number of bytes for the package version.
- `uint8_t packageName [ICP_DCC_COMPONENT_NAME_LENGTH]`
Package name as per Version Information structure.
- `uint8_t packageVersion [ICP_DCC_VERSION_STR_LENGTH]`
Package version information in the form of a string.

3.5.2.3 Field Documentation

`uint8_t icp_dcc_package_ver_s::packageSize`

Number of bytes for the package version.

`uint8_t icp_dcc_package_ver_s::packageName[ICP_DCC_COMPONENT_NAME_LENGTH]`

Package name as per Version Information structure.

`uint8_t icp_dcc_package_ver_s::packageVersion[ICP_DCC_VERSION_STR_LENGTH]`

Package version information in the form of a string.

3.5.3 icp_dcc_module_ver_s Struct Reference

3.5.3.1 Detailed Description

Structure of Module version in buffer returned to Debug Client.

This structure contains the Module version information. This information is returned in the buffer provided by the Debug Client.

Purpose:

When queried for software version, the version information of the registered Modules are returned in this format.

3.5.3.2 Data Fields

- `uint8_t componentSize`
Number of bytes for the component version.
- `uint32_t moduleId`
Module ID.
- `uint8_t moduleName [ICP_DCC_COMPONENT_NAME_LENGTH]`
Module name as per Version Information structure.

3.5.3 icp_dcc_module_ver_s Struct Reference

- uint8_t **moduleVersion** [ICP_DCC_VERSION_STR_LENGTH]
Module version information in the form of a string.

3.5.3.3 Field Documentation

uint8_t **icp_dcc_module_ver_s::componentSize**

Number of bytes for the component version.

uint32_t **icp_dcc_module_ver_s::moduleId**

Module ID.

uint8_t **icp_dcc_module_ver_s::moduleName**[ICP_DCC_COMPONENT_NAME_LENGTH]

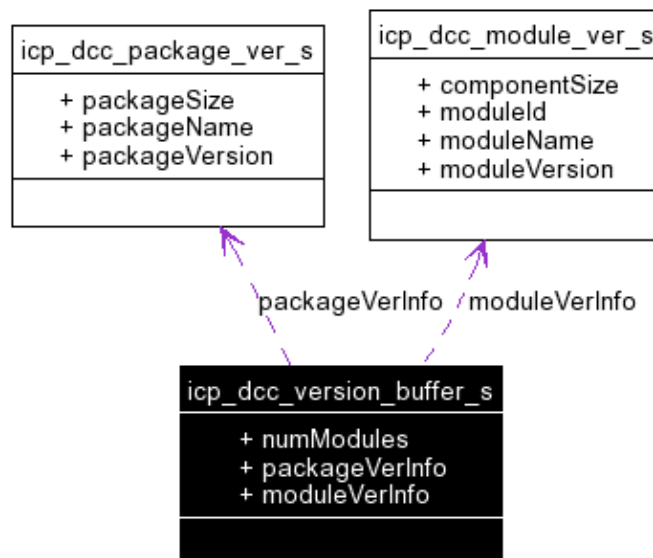
Module name as per Version Information structure.

uint8_t **icp_dcc_module_ver_s::moduleVersion**[ICP_DCC_VERSION_STR_LENGTH]

Module version information in the form of a string.

3.5.4 icp_dcc_version_buffer_s Struct Reference

Collaboration diagram for icp_dcc_version_buffer_s:



3.5.4.1 Detailed Description

Structure of Version buffer returned to Debug Client.

This structure provides Package/Module version information in the buffer provided by Debug Client.

Purpose:

When queried for software version, the version information of the Package, as well as all registered Modules is returned in this format.

3.5.4 icp_dcc_version_buffer_s Struct Reference

3.5.4.2 Data Fields

- **uint32_t numModules**
Count of number of module version fields within the buffer.
- **icp_dcc_package_ver_t packageVerInfo**
Package version information as provided to client.
- **icp_dcc_module_ver_t moduleVerInfo []**
Module version information as provided to client.

3.5.4.3 Field Documentation

uint32_t icp_dcc_version_buffer_s::numModules

Count of number of module version fields within the buffer.

icp_dcc_package_ver_t icp_dcc_version_buffer_s::packageVerInfo

Package version information as provided to client.

icp_dcc_module_ver_t icp_dcc_version_buffer_s::moduleVerInfo[]

Module version information as provided to client.

3.6 Typedef Documentation

typedef struct icp_dcc_liveness_status_s icp_dcc_liveness_status_t

Thread response Status .

This value denotes the response status of the specified thread ID.

Purpose:

During response check, the response status of each thread is returned through this structure.

typedef struct icp_dcc_package_ver_s icp_dcc_package_ver_t

Structure of Package version in buffer returned to Debug Client.

This structure provides Package version information in buffer provided by the Debug Client.

Purpose:

When queried for software version, the version information of the Package is returned in this format.

typedef struct icp_dcc_module_ver_s icp_dcc_module_ver_t

Structure of Module version in buffer returned to Debug Client.

This structure contains the Module version information. This information is returned in the buffer provided by the Debug Client.

Purpose:

When queried for software version, the version information of the registered Modules are returned in this format.

typedef struct icp_dcc_version_buffer_s icp_dcc_version_buffer_t

3.6 Typedef Documentation

Structure of Version buffer returned to Debug Client.

This structure provides Package/Module version information in the buffer provided by Debug Client.

Purpose:

When queried for software version, the version information of the Package, as well as all registered Modules is returned in this format.

```
typedef void(* icp_DccSenHandler)(icp_dcc_sen_msg_t *pSenMsg)
```

Prototype of SEN event callback handler to be registered by user.

The DCC user registers the SEN event callback handler with this function. When a SEN event occurs, the DCC calls the handler registered in this prototype format.

Context:

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

Yes

Thread-safe:

Yes

Parameters:

pSenMsg,: IN/OUT: SEN Event Message Details

Return values:

ICP_STATUS_SUCCESS Operation successful
ICP_STATUS_FAIL Operation failed
ICP_STATUS_NULL_PARAM One or more parameters are NULL

Precondition:

None

Postcondition:

None

See also:

3.7 Function Documentation

```
icp_status_t icp_DccVersionInfoSizeGet ( uint32_t * pVersionBufferSize )
```

Get the maximum buffer size needed to retrieve Version information of the Package and all the Components registered with the DCC.

3.7 Function Documentation

This function provides the DCC Client with buffer size needed to retrieve Version information of all the Package/Components registered with DCC. The DCC user should allocate this buffer and pass it to DCC to retrieve the Package and Components version information.

Context:

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is synchronous and blocking.

Reentrant:

Yes

Thread-safe:

Yes

Parameters:

pVersionBufferSize,: IN/OUT: DCC Package and Component information

Return values:

ICP_STATUS_SUCCESS Operation successful
ICP_STATUS_FAIL Operation failed
ICP_STATUS_NULL_PARAM One or more parameters are NULL

Precondition:

None

Postcondition:

None

```
icp_status_t icp_DccSoftwareVersionGet (
    uint8_t * pVerInfo,
    uint32_t * pVersionBufferSize
)
```

Returns Package and Components version information.

This function provides the caller with version information for the Package and each of the Components registered with the DCC.

Context:

This function runs in the context of the calling function thread.

Assumptions:

This function should be called with the required amount of buffer allocated for the returned information.

Side-Effects:

None

3.7 Function Documentation

Blocking:

This function is synchronous and blocking.

Reentrant:

Yes

Thread-safe:

Yes

Parameters:

pVerInfo,: IN/OUT: The array containing the version information for the Package and for all the components registered with the DCC. This is a pointer to the array of "icp_dcc_version_buffer_t".

pVersionBufferSize,: IN/OUT: While calling this API, the user specifies the amount of memory allocated which can be used by the DCC to populate the version information for all the components registered with the DCC. The DCC checks if the buffer size specified can accommodate the version information retrieved. If so, the DCC fills in the component version information for passing back to the user. Typically, the size should be the same as that provided when the user calls the "icp_DccVersionInfoSizeGet" function. In the returned information, the first data corresponds to Package information and subsequent data corresponds to individual component versions.

Return values:

ICP_STATUS_SUCCESS Operation successful
ICP_STATUS_FAIL Operation failed
ICP_STATUS_NULL_PARAM One or more parameters are NULL

Precondition:

None

Postcondition:

None

See also:

icp_DccVersionInfoSizeGet

icp_dcc_version_buffer_t

icp_status_t icp_DccLivenessConfigureTimeout (uint32_t *livenessTimeOut*)

Set the response timeout value for DCC.

This API is used to configure the timeout period (in ms) for response monitoring. The DCC waits for a response during this period before declaring a thread to be alive or dead. The timeout period should be large enough to allow all threads to respond appropriately within this period.

Context:

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

3.7 Function Documentation

Blocking:

This function is blocking.

Reentrant:

Yes

Thread-safe:

Yes

Parameters:

livenessTimeout,: IN: Timeout value

Return values:

ICP_STATUS_SUCCESS Operation successful

ICP_STATUS_FAIL Operation failed

Precondition:

None

Postcondition:

None

icp_status_t icp_DccLivenessResponseSizeGet (uint32_t * *pLivenessResponseSize*)

Get the buffer size to be allocated by the DCC user for getting the system response information.

The DCC user uses this API to get the buffer size to be allocated for the retrieval of system response information. The DCC user should free this buffer after the buffer information is processed.

Context:

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

Yes

Thread-safe:

Yes

Parameters:

pLivenessResponseSize,: IN/OUT: The buffer size to be allocated while quering the response status

Return values:

ICP_STATUS_SUCCESS Operation successful

ICP_STATUS_FAIL Operation failed

Precondition:

None

3.7 Function Documentation

Postcondition:

None

```
icp_status_t icp_DccLivenessVerify ( uint8_t* pLivenessStatus,  
                                     uint32_t* pBufferSize  
                                     )
```

Verify System response.

The DCC user uses this API to verify response of all threads of execution in the system. The user provides the required buffer which the DCC fills with Response status information.

Context:

This function runs in the context of the calling function thread. This should not be called in interrupt mode.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

Yes

Thread-safe:

Yes

Parameters:

pLivenessStatus,: IN/OUT: The system threads execution status. This is a pointer to the array of structure `icp_dcc_liveness_status_t`.

pBufferSize,: IN/OUT: The buffer size allocated to return the system response status.

Return values:

`ICP_STATUS_SUCCESS` Operation successful

`ICP_STATUS_FAIL` Operation failed

`ICP_STATUS_INVALID_PARAM` Invalid parameter passed

`ICP_STATUS_NULL_PARAM` One or more parameters are NULL

Precondition:

None

Postcondition:

None

See also:

`icp_dcc_liveness_status_t`

`icp_DccLivenessResponseSizeGet`

```
icp_status_t icp_DccDataDumpInfoGet ( uint32_t* noOfModules,  
                                       uint32_t* pDataDumpSize  
                                       )
```

3.7 Function Documentation

Get the number of Modules and the maximum buffer size required to get a data dump.

This function returns the number of Modules and the maximum size of buffer needed for any one dump query transaction. Before invoking the Data Dump query request, the DCC user gets the maximum size of the buffer to allocate and the total number of Modules that should be queried for the data dump.

Context:

This function runs in the context of the calling function thread. This function should not be called from an interrupt context.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

No

Thread-safe:

Yes

Parameters:

noOfModules,: IN/OUT: Number of Modules registered at DCC for Data Dump
pDataDumpSize,: IN/OUT: Maximum buffer size required to query any one dump Module

Return values:

<i>ICP_STATUS_SUCCESS</i>	Operation successful
<i>ICP_STATUS_FAIL</i>	Operation failed
<i>ICP_STATUS_INVALID_PARAM</i>	Invalid parameter passed
<i>ICP_STATUS_NULL_PARAM</i>	One or more parameters are NULL

Precondition:

None

Postcondition:

None

See also:

```
icp_status_t icp_DccDataDumpGet ( uint32_t  moduleId,  
                                  uint32_t*  pDataDumpSize,  
                                  uint8_t*   pDataDump  
                                  )
```

Get the number of Modules and the maximum buffer size required to get a data dump.

The User should use this function to dump the data structures of Modules registered with the DCC. This function will be called once for each registered Module for the data dump information. With each call, the user passes to DCC a pre-allocated buffer which is passed to the Module to be filled with the dump information.

Context:

3.7 Function Documentation

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

No

Thread-safe:

Yes

Parameters:

moduleId,: IN: Opaque index representing the Module being queried for data dump
pDataDump,: IN/OUT: Data Dump information from the Module
pDataDumpSize,: IN/OUT: Actual size of dump data present in the buffer

Return values:

ICP_STATUS_SUCCESS Operation successful
ICP_STATUS_FAIL Operation failed
ICP_STATUS_INVALID_PARAM Invalid parameter passed
ICP_STATUS_NULL_PARAM One or more parameters are NULL

Precondition:

None

Postcondition:

None

See also:

icp_DccDataDumpInfoGet

icp_status_t icp_DccSenHandlerRegister (**icp_DccSenHandler** *senHandler*)

Register the SEN event callback handler.

The DCC user registers the SEN event callback handler with this function.

Context:

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

3.7 Function Documentation

No

Thread-safe:

Yes

Parameters:

senHandler IN: SEN Handler callback to be registered

Return values:

ICP_STATUS_SUCCESS Operation successful

ICP_STATUS_FAIL Operation failed

Precondition:

None

Postcondition:

None

See also:

icp_DccSenHandler

icp_status_t icp_DccSenHandlerUnregister (void)

Unregister the SEN event callback handler.

The DCC user unregisters the SEN event callback handler.

Context:

This function runs in the context of the calling function thread.

Assumptions:

None

Side-Effects:

None

Blocking:

This function is blocking.

Reentrant:

No

Thread-safe:

Yes

Parameters:

@retval *ICP_STATUS_SUCCESS* Operation successful

Return values:

ICP_STATUS_FAIL Operation failed

Precondition:

None

Postcondition:

3.7 Function Documentation

None

See also:

4 Debug Base Data Structures

4.1 Detailed Description

This section describes the data structures, defines, typedefs and enumerations used by acceleration software components and Debug Clients.

4.2 Data Structures

- struct **icp_dcc_thread_id_s**
Thread ID.
- struct **icp_dcc_ver_info_s**
Software Version information.
- struct **icp_dcc_sen_msg_s**
Software Error Notification (SEN) event message details.

4.3 Defines

- #define **ICP_DCC_MAX_MODULES**
Maximum number of Modules supported by the DCC.
- #define **ICP_DCC_MAX_THREADS**
Maximum number of threads supported by the DCC.
- #define **ICP_DCC_THREAD_ID_LENGTH**
Maximum Length of Thread ID.
- #define **ICP_DCC_COMPONENT_NAME_LENGTH**
Maximum Length of Package/Component name.
- #define **ICP_DCC_MAX_MSG_SIZE**
Length of the SEN Message.
- #define **ICP_DCC_MODULE_ID_STR_LENGTH**
Length of Module ID String.
- #define **ICP_DCC_VERSION_STR_LENGTH**
Length of Version String.

4.4 Typedefs

- typedef enum **icp_dcc_thread_status_s icp_dcc_thread_status_t**
Response state of the Thread.
- typedef **icp_dcc_thread_id_s icp_dcc_thread_id_t**
Thread ID.
- typedef **icp_dcc_ver_info_s icp_dcc_ver_info_t**
Software Version information.
- typedef enum **icp_dcc_sen_priority_s icp_dcc_sen_priority_t**
Priority of the Software Error Notification (SEN) event message.
- typedef **icp_dcc_sen_msg_s icp_dcc_sen_msg_t**
Software Error Notification (SEN) event message details.

4.5 Enumerations

- enum **icp_dcc_thread_status_s** {
 ICP_DCC_THREAD_ID_LIVE,
 ICP_DCC_THREAD_ID_DEAD

4.5 Enumerations

- ```
}
 Response state of the Thread.
• enum icp_dcc_sen_priority_s {
 ICP_DCC_SEN_MSG_PRIORITY_SEVERE,
 ICP_DCC_SEN_MSG_PRIORITY_WARNING
}
```
- Priority of the Software Error Notification (SEN) event message.
- 

## 4.6 Data Structure Documentation

### 4.6.1 icp\_dcc\_thread\_id\_s Struct Reference

#### 4.6.1.1 Detailed Description

Thread ID.

This structure provides thread Identification details.

**Purpose:**

This structure provides Thread ID as a string.

#### 4.6.1.2 Data Fields

- `uint8_t threadIdString` [ICP\_DCC\_THREAD\_ID\_LENGTH]  
Thread ID String.

#### 4.6.1.3 Field Documentation

```
uint8_t icp_dcc_thread_id_s::threadIdString[ICP_DCC_THREAD_ID_LENGTH]
 Thread ID String.
```

---

### 4.6.2 icp\_dcc\_ver\_info\_s Struct Reference

#### 4.6.2.1 Detailed Description

Software Version information.

This structure provides Software Version information.

**Purpose:**

The Software Version information of Package and Components are provided through this structure.

#### 4.6.2.2 Data Fields

- `uint8_t name` [ICP\_DCC\_COMPONENT\_NAME\_LENGTH]  
Software Package/Component Name.
- `uint8_t majorVersion`  
Software Major Version Number.
- `uint8_t minorVersion`  
Software Minor Version Number.
- `uint16_t patchVersion`  
Software Patch Version Number.

## 4.6.2 icp\_dcc\_ver\_info\_s Struct Reference

### 4.6.2.3 Field Documentation

uint8\_t **icp\_dcc\_ver\_info\_s::name**[ICP\_DCC\_COMPONENT\_NAME\_LENGTH]

Software Package/Component Name.

uint8\_t **icp\_dcc\_ver\_info\_s::majorVersion**

Software Major Version Number.

uint8\_t **icp\_dcc\_ver\_info\_s::minorVersion**

Software Minor Version Number.

uint16\_t **icp\_dcc\_ver\_info\_s::patchVersion**

Software Patch Version Number.

---

## 4.6.3 icp\_dcc\_sen\_msg\_s Struct Reference

### 4.6.3.1 Detailed Description

Software Error Notification (SEN) event message details.

This structure contains SEN event details.

#### Purpose:

This data structure defines the SEN message structure. This message structure has to be formed by the Acceleration Software Modules and sent to DCC. DCC passes this information to the Debug Client's registered event handler.

### 4.6.3.2 Data Fields

- **icp\_dcc\_sen\_priority\_t senPriority**  
Priority of the SEN event.
- **uint64\_t timestamp**  
Timestamp when the event is received by DCC.
- **uint32\_t moduleId**  
Module ID.
- **uint16\_t eventId**  
Event ID.
- **uint16\_t eventInfoSize**  
Size of the SEN eventInfo string in bytes.
- **uint8\_t eventInfo** [ICP\_DCC\_MAX\_MSG\_SIZE]  
Descriptive null-terminated SEN Event string.

### 4.6.3.3 Field Documentation

**icp\_dcc\_sen\_priority\_t icp\_dcc\_sen\_msg\_s::senPriority**

Priority of the SEN event.

**uint64\_t icp\_dcc\_sen\_msg\_s::timestamp**

Timestamp when the event is received by DCC.

### 4.6.3 icp\_dcc\_sen\_msg\_s Struct Reference

uint32\_t **icp\_dcc\_sen\_msg\_s::moduleId**

Module ID.

uint16\_t **icp\_dcc\_sen\_msg\_s::eventId**

Event ID.

uint16\_t **icp\_dcc\_sen\_msg\_s::eventInfoSize**

Size of the SEN eventInfo string in bytes.

uint8\_t **icp\_dcc\_sen\_msg\_s::eventInfo**[ICP\_DCC\_MAX\_MSG\_SIZE]

Descriptive null-terminated SEN Event string.

---

## 4.7 Define Documentation

**#define ICP\_DCC\_MAX\_MODULES**

Maximum number of Modules supported by the DCC.

The number of Modules that shall register their version information with DCC, apart from the Package version information.

**Purpose:**

The Maximum number of Modules that shall be supported by DCC.

**#define ICP\_DCC\_MAX\_THREADS**

Maximum number of threads supported by the DCC.

The number of threads registered with the DCC shall not exceed this value.

**Purpose:**

Threads register with the DCC to provide their responsiveness. The number of threads registered with DCC at any point in time should not exceed this value.

**#define ICP\_DCC\_THREAD\_ID\_LENGTH**

Maximum Length of Thread ID.

This value specifies the length of the Thread ID.

**Purpose:**

The Thread ID is defined as a character string. The length of the string is defined by this value.

**#define ICP\_DCC\_COMPONENT\_NAME\_LENGTH**

Maximum Length of Package/Component name.

This value specifies the length of the component name.

**Purpose:**

The component name is defined as a character string. The length of the string is defined by this value.

**#define ICP\_DCC\_MAX\_MSG\_SIZE**

## 4.7 Define Documentation

Length of the SEN Message.

The Message length should not exceed this value.

**Purpose:**

SEN messages are defined as character strings. The maximum size of this string is defined by this value.

```
#define ICP_DCC_MODULE_ID_STR_LENGTH
```

Length of Module ID String.

The Module ID string length should not exceed this value.

**Purpose:**

The Module ID, when converted to string format is defined as a character string. The maximum length of this string is defined by this value.

```
#define ICP_DCC_VERSION_STR_LENGTH
```

Length of Version String.

The Version string length shall not exceed this value.

**Purpose:**

The Version String is obtained from major version, minor version and patch version. The maximum size of this string is defined by this value.

---

## 4.8 Typedef Documentation

```
typedef enum icp_dcc_thread_status_s icp_dcc_thread_status_t
```

Response state of the Thread.

This structure provides the thread state. Possible states are "live" or "dead".

**Purpose:**

This enumeration type defines the possible states of the executing thread. It indicates whether the thread has responded back within a certain time period or not.

```
typedef struct icp_dcc_thread_id_s icp_dcc_thread_id_t
```

Thread ID.

This structure provides thread Identification details.

**Purpose:**

This structure provides Thread ID as a string.

```
typedef struct icp_dcc_ver_info_s icp_dcc_ver_info_t
```

Software Version information.

This structure provides Software Version information.

**Purpose:**

## 4.8 Typedef Documentation

The Software Version information of Package and Components are provided through this structure.

```
typedef enum icp_dcc_sen_priority_s icp_dcc_sen_priority_t
Priority of the Software Error Notification (SEN) event message.
```

This structure provides supported SEN message priorities.

**Purpose:**

This enumeration type defines the priorities of the SEN events that occur in the system. Supported priority types are: Severe and Warning.

```
typedef struct icp_dcc_sen_msg_s icp_dcc_sen_msg_t
Software Error Notification (SEN) event message details.
```

This structure contains SEN event details.

**Purpose:**

This data structure defines the SEN message structure. This message structure has to be formed by the Acceleration Software Modules and sent to DCC. DCC passes this information to the Debug Client's registered event handler.

---

## 4.9 Enumeration Type Documentation

```
enum icp_dcc_thread_status_s
Response state of the Thread.
```

This structure provides the thread state. Possible states are "live" or "dead".

**Purpose:**

This enumeration type defines the possible states of the executing thread. It indicates whether the thread has responded back within a certain time period or not.

**Enumerator:**

*ICP\_DCC\_THREAD\_ID\_LIVE* Thread is responding.  
*ICP\_DCC\_THREAD\_ID\_DEAD* Thread is not responding.

```
enum icp_dcc_sen_priority_s
Priority of the Software Error Notification (SEN) event message.
```

This structure provides supported SEN message priorities.

**Purpose:**

This enumeration type defines the priorities of the SEN events that occur in the system. Supported priority types are: Severe and Warning.

**Enumerator:**

*ICP\_DCC\_SEN\_MSG\_PRIORITY\_SEVERE* Critical errors impacting functionality or requiring immediate attention.  
*ICP\_DCC\_SEN\_MSG\_PRIORITY\_WARNING* Important system notifications.