OpenClovis Software Development Kit (SDK) Service Description and API Reference for Intelligent Object Communication (IOC) Service

For OpenClovis SDK Release 2.3 V0.4

Document Revision Date: March 30, 2007

Contents

Chapter 1

Functional Overview

TBD

Chapter 2

Service Model

TBD

Chapter 3

Service APIs

3.1 Type Definitions

3.1.1 ClIocPortT

typedef ClUint32T ClIocPortT;

The data type of the IOC communication port. This is the physical address of the port created for communication within a node.

3.1.2 ClIocCommPortFlagsT

typedef ClUint32T ClIocCommPortFlagsT;

The type of a flag of the IOC communication port. It can be used to specify the attributes of the type of communication you need to use. Currently unreliable messaging is supported:
`CL_IOC_UNRELIABLE_MESSAGING` - for unreliable messaging.

3.1.3 ClIocCommPortHandleT

typedef ClHandleT ClIocCommPortHandleT;

The type of the handle of the IOC communication port. After the port is opened all the operations to be performed on the port are performed on the handle.

3.1.4 ClIocSendOptionT

```
typedef struct {
        ClIocToHandleT toHandle;
        ClIocMessageOptionT msgOption;
        ClUint8T priority;
        ClUint8T sendType;
        ClUint32T timeout;
} ClIocSendOptionT;
```

The structure, `ClIocSendOptionT`, contains the various options for the `clIocSend()` function. The options are:

- *msgOption* - Type of the message. This can be persistent or non-persistent and can have any of the following two values:
  - `CL_IOC_PERSISTENT_MSG`: For persistent message, user is expected to free the message
  - `CL_IOC_NON_PERSISTENT_MSG`: Default value. The message is freed by IOC
- *priority* - Priority of the message being sent. The range of values it can have is from 1 to the maximum value configured at the time of initialization. If it is greater than the maximum supported value, the message is sent with the least priority.
- *sendType* - Used to maintain a session. It can have one of the following values:
  - `CL_IOC_SESSION_BASED` : If user wants to maintain a session
  - `CL_IOC_NO_SESSION` : Default value
- *timeout* - Timeout interval in milliseconds.
- *toHandle* - Handle used to uniquely identify the transport.

3.1.5 ClIocRecvParamT

```
typedef struct {
        ClUint32T length;
        ClUint8T priority;
        ClUint8T protoType;
        ClIocAddressT srcAddr;
} ClIocRecvParamT;
```

This structure, `ClIocRecvParamT`, contains the parameters of the received message. It is returned by the `clIocReceive()` function with the message. The attributes of the structure are:

- *length* - Length of the received message.
- *priority* - Priority of the received message.
- *protoType* - Protocol used.
- *srcAddr* - Address of the sender.

3.1.6 ClIocRecvOption

```
typedef struct {
        ClUint32T recvTimeout;
} ClIocRecvOptionT;
```

The structure, `ClIocRecvOptionT`, contains the IOC receive options that can be used to customize the receive call (`clIocReceive()`). Currently, timeout value for blocking call is supported.

- *recvTimeout* - Time-out value for the blocking receive call. The receive call is blocked till it receives some data or till it times out. The time-out value is measured in milliseconds. The default value is `CL_IOC_TIMEOUT_FOREVER`.

3.1.7 ClIocTLInfoT

typedef struct ClIocTlInfo{
ClIocLogicalAddressT logicalAddr;
ClUint32T compId;
ClIocTLContextT contextType;
ClIocTLReplicationT repliSemantics;
ClUint32T haState;
ClIocPhysicalAddressT physicalAddr;
} ClIocTLInfoT;

The structure, `ClIocTlInfo`, contains the information required to create an entry in the transparency layer. The attributes of this structure are:

- *logicalAddr* - Logical address of the service.
- *compId* - ID of the component providing the service.
- *contextType* - Type of the context. It can be either `GLOBAL` or `LOCAL`.
- *repliSemantics* - Replication semantics.
- *haState* - Active or Standby mode.
- *physicalAddr* - Physical address of the component.

3.1.8 ClIocLogicalAddressT

typedef ClUint64T ClIocLogicalAddressT;

The type of the IOC Logical address.

3.1.9 ClIocTLMappingT

typedef struct {
ClUint32T haState;
ClIocPhysicalAddressT physicalAddr;
} ClIocTLMappingT;

The structure, `ClIocTLMappingT`, contains the physical address and its state. The physical address corresponds to a logical address in the transparency layer. The attributes of this structure are:

- *haState* - State of the component corresponding to `physicalAddr`.
- *physicalAddr* - Physical address of the component.

3.1.10 ClIocToBindHandleT

typedef ClHandleT ClIocToBindHandleT;

The type of the handle for transport object. It is used to send a packet using a specific transport object.

3.1.11 ClIocRouteParamT

```
typedef struct ClIocRouteParam{
        ClIocNodeAddressT destAddr;
        ClIocNodeAddressT nextHop;
        ClUint16T prefixLen;
        ClUint16T metrics;
        ClCharT *pXportName;
        ClCharT *pLinkName;
        ClUint8T flags;
        ClUint8T version;
        ClUint8T status;
        ClUint8T entryType;
} ClIocRouteParamT;
```

The structure, `ClIocRouteParamT`, contains the information required to add a route. The attributes of this structure are:

- *destAddr* - Address of the destination IOC .
- *nextHop* - Next hop IOC address to reach the destination address.
- *prefixLen* - Length of the prefix.
- *metrics* - Metrics of the route.
- *pXportName* - Name of the transport, selected to reach the destination address.
- *pLinkName* - Name of the link selected to reach the destination address. This link should belong to the transport, identified by `pXportName`.
- *flags* - Type of the route .
- *version* - Version of IOC on next hop.
- *status* - Status of the route .
- *entryType* - Type of the route. It can be static or dynamic.

3.1.12 ClIocTransportConfigT

```
typedef struct ClIocTransportConfig{
        ClUint8T version;
        ClCharT pXportName [CL_IOC_MAX_XPORT_NAME_LENGTH + 1];
        ClUint8T priority;
        ClUint8T xportType;
        ClIocTransportFuncT initRoutine;
```

```
        ClIocTransportSendFuncT sendRoutine;
        ClIocTransportFuncT closeRoutine;
        ClIocTransportAddrConvertFuncT addrConvertRoutine;
        ClIocTransportAddrConvertFuncT addrExtractRoutine;
} ClIocTransportConfigT;
```

The structure, `ClIocTransportConfigT`, contains the configuration information of the transport object. The attributes of this structure are:

- *version* - Version of the IOC.
- *pXportName* - Name of the transport.
- *priority* - Type of the transport used for communication.
- *xportType* - Type of the transport used for communication.
- *initRoutine* - Initialization routine for the transport.
- *sendRoutine* - Send routine of the user transport. This is called when data is to be sent on the transport.
- *closeRoutine* - Finalization routine to close the transport.
- *addrConvertRoutine* - Address conversion routine.
- *addrExtractRoutine* - Address extraction routine.

3.1.13 ClIocAddressT

```
typedef union ClIocAddress{
        ClIocPhysicalAddressT iocPhyAddress;
        ClIocLogicalAddressT iocLogicalAddress;
        ClIocMulticastAddressT iocMulticastAddress;
} ClIocAddressT;
```

The type of the IOC address. The types of addresses can be physical address, logical address, multicast address, and broadcast address.

- *iocPhyAddress* - Physical address.
- *iocLogicalAddress* - Logical address.
- *iocMulticastAddress* - Multicast address.

3.1.14 ClIocNodeAddressT

```
typedef ClUint32T ClIocNodeAddressT;
```

The type of an identifier for the IOC node address.

3.1.15 ClIocArpParamT

```
typedef struct ClIocArpParam{
        ClIocNodeAddressT iocAddr;
        ClUint8T *pTransportAddr;
        ClUint32T addrSize;
        ClCharT *pXportName;
        ClCharT *pLinkName;
        ClUint8T status;
        ClUint8T entryType;
} ClIocArpParamT;
```

The structure, `ClIocArpParamT`, contains the information to add IOC ARP entries. This ARP is for the IOC physical blade address to link with the address resolution. The information to add IOC ARP entries include:

- *iocAddr* - The IOC address whose ARP entry is to be added.
- *pTransportAddr* - Transport address (link address) of the IOC.
- *addrSize* - Size of the address.
- *pXportName* - Name of the transport to which the entry is associated.
- *pLinkName* - Name of the link to which the entry is associated. This link should belong to the transport identified by `pXportName`.
- *status* - Status of the remote node on this transport and link.
- *entryType* - Type of the entry. It can be static or dynamic ARP.

3.1.16 ClIocTransportLinkConfigT

```
typedef struct ClIocTransportLinkConfig{
        ClCharT pXportName [CL_IOC_MAX_XPORT_NAME_LENGTH + 1];
        ClCharT pXportLinkName [CL_IOC_MAX_XPORT_NAME_LENGTH + 1];
        ClUint8T xportType;
        ClUint8T isChecksumReqd;
        ClUint8T addressSize;
        ClUint8T isBcastSupported;
        ClUint8T xportBcastAddress [CL_IOC_MAX_XPORT_NAME_LENGTH + 1];
        ClUint8T xportAddress [CL_IOC_MAX_XPORT_NAME_LENGTH + 1];
        ClUint32T mtuSize;
        ClIocTransportStatsT *pIocXportStats;
        ClIocCoreFuncT iocCoreRecvRoutine;
        ClUint8T priority;
        ClUint8T isRegistered;
        void *pXportLinkPrivData;
        ClUint8T status;
} ;
```

The structure, `ClIocTransportLinkConfigT`, contains the attributes required to configure a link in a transport object. They include:

- *pXportName* - Transport Name on which the link is present.
- *pXportLinkName* - Name of the link.
- *xportType* - Type of the transport used for communication.
- *isChecksumReqd* - Indicates if Checksum support is enabled or disabled.
- *addressSize* - Size of the address of the link.
- *isBcastSupported* - Indicates if broadcast support is present.
- *xportBcastAddress* - Broadcast address.
- *xportAddress* - Transport address.
- *mtuSize* - Maximum size of messages that can be transmitted. It must be more than `CL_IOC_MIN_MTU_SIZE`.
- *pIocXportStats* - Statistics of transport.
- *iocCoreRecvRoutine* - This routine is called every time you pass data to IOC.
- *priority* - Priority of the link.
- *isRegistered* - Link registration information (not required to be passed by the user).
- *pXportLinkPrivData* - Private information related to the link.
- *status* - Status of the link (not required to be passed by the user).

3.1.17 ClIocUserTransportConfigT

```
typedef struct ClIocXportConfig{
        ClCharT pName[CL_IOC_MAX_XPORT_NAME_LENGTH + 1];
        ClUint8T priority;
        ClUint32T numOfLinks;
        ClIocUserLinkCfgT *pLink;
} ClIocUserTransportConfigT;
```

The structure, `ClIocUserTransportConfigT`, contains the configuration information of the transport. The attributes of this structure are:

- *pName* - Name of the transport.
- *priority* - Type of the transport.(not required to be passed by the user).
- *numOfLinks* - Total number of links on this transport.
- *pLink* - Configuration information of this link. If there is more than one link, pass a pointer to an array of the `ClIocUserLinkCfgT` structure.

3.1.18 ClIocQueueStatsT

typedef struct ClIocQueueStats {
ClWaterMarkT queueWaterMark;
ClUint32T queueSize;
ClUint32T queueUtilisation;
} ClIocQueueStatsT;

The structure, `ClIocQueueStatsT`, contains the statistics of the IOC queue. The attributes of this structure are:

- *queueWaterMark* - Low and high watermark limits for the IOC queues. This is a percentage of `queueSize`.
- *queueSize* - Maximum queue size in bytes.
- *queueUtilisation* - Current queue utilization in bytes.

3.1.19 ClIocQueueIdT

typedef enum ClIocQueueId {
CL_IOC_SEND_QUEUE,
CL_IOC_COMMPORT_RECV_QUEUE,
} ClIocQueueIdT;

The enumeration, `ClIocQueueIdT`, contains the IDs of the IOC queue. The attributes of this structure are:

- *CL_IOC_SEND_QUEUE* - IOC send queue, which is one per node.
- *CL_IOC_COMMPORT_RECV_QUEUE* - IOC communication port receive queue, which is one per communication port.

3.2 Functional APIs

3.2.1 clIocCommPortCreate

clIocCommPortCreate

Synopsis:
Creates an IOC communication port.

Header File:
clIocApi.h

Syntax:

```
ClRcT clIocCommPortCreate(
                          CL_IN  ClIocPortT portId,
                          CL_IN  ClIocCommPortFlagsT portType,
                          CL_OUT ClIocCommPortHandleT *pIocCommPortHdl);
```

Parameters:

portId: (in) ID of the communication port to be created. If `portId` is 0, a `portId` is generated by IOC.

portType: (in) Type of communication that can be either reliable or unreliable. This parameter can have the following two values:

- `CL_IOC_UNRELIABLE_MESSAGING` - For unreliable messaging.
- `CL_IOC_RELIABLE_MESSAGING` - For reliable messaging.

Currently, unreliable messaging is supported.

plocCommPortHdl: (out) Handle to the communication port used by applications to send and receive the messages.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pIocCommPortHdl` is NULL.

CL_ERR_NOT_IMPLEMENTED: `portType` is not set to `CL_IOC_UNRELIABLE_MESSAGING`.

CL_ERR_INVALID_PARAMETER: `portId` is greater than `CL_IOC_COMMPORT_END`.

CL_ERR_NOT_EXIST: `portId` is zero and no ephemeral communication port is free.

CL_IOC_ERR_COMMPORT_REG_FAIL: Communication port registration failed.

CL_ERR_NO_MEMORY: Memory allocation or any other resource allocation failed.

CL_ERR_UNSPECIFIED: An unexpected failure has occurred.

Description:
This function is used to create a communication port, which is used to send and receive data. This function needs to be invoked before any communication port related operations. The communication type can be reliable or unreliable. OpenClovis ASP supports only unreliable mode of communication. After creation, the communication port is in the blocking mode.

Library File:
CIIoc

Related Function(s):
clIocSend, clIocReceive, clIocCommPortDelete

3.2.2 clIocCommPortDelete

clIocCommPortDelete

Synopsis:
Deletes the IOC communication port.

Header File:
clIocApi.h

Syntax:
```
ClRcT clIocCommPortDelete(
                      CL_IN ClIocCommPortHandleT iocCommPortHdl);
```

Parameters:
iocCommPortHdl: (in) Handle of the communication port to be deleted.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: If the IOC is not initialized

CL_ERR_INVALID_HANDLE: Communication port handle is invalid.

CL_IOC_ERR_COMMPORT_BLOCKED: Communication port is blocked.

Description:
This function is used to delete the communication port created using the `clIocCommPortCreate()` function. The communication port cannot be deleted, if it is being used by a thread or, if it is blocked for receive. If a thread is blocked, the error, `CL_IOC_ERR_COMMPORT_BLOCKED` is returned. When this function is successfully executed, communication port becomes invalid.

Library Files:
Cllog

Related Function(s):
clIocCommPortCreate, clIocSend, clIocReceive

3.2.3 clIocSend

clIocSend

Synopsis:
Sends messages to a communication port.

Header File:
clIocApi.h

Syntax:

```
ClRcT clIocSend(
                  CL_IN ClIocCommPortHandleT commPortHandle,
                  CL_IN ClBufferHandleT message,
                  CL_IN ClUint8T protoType,
                  CL_IN ClIocAddressT* pDestAddr,
                  CL_IN ClIocSendOptionT* pSendOption);
```

Parameters:

commPortHandle: (in) Handle to a communication port on which a message is to be sent.

message: (in) Message to be sent across the communication port. This message must be created by the user. If the message is persistent, it must be freed by the user.

protoType: (in) Protocol ID specified by the user.

pDestAddr: (in) Pointer to the destination address where the message needs to be sent.

pSendOption: (in) Options available to send a message. If `pSendOption` is NULL, the default values are used. For more information, refer the structure, `ClIocSendOptionT`, in the Type Definitions chapter.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_HANDLE: Communication port handle is invalid.

CL_ERR_NULL_POINTER: `pDestAddr` is NULL.

CL_ERR_INVALID_BUFFER: Message is invalid.

CL_IOC_ERR_PROTO_IN_USE_WITH_IOC: Protocol ID is already in use by IOC.

CL_IOC_ERR_INVALID_MSG_OPTION: Message option passed in `pSendOption` is invalid.

CL_ERR_INVALID_PARAMETER: The `sendType` passed in the `pSendOption` is invalid or the destination address is not of supported type, or the message size is 0.

CL_ERR_NO_MEMORY: Memory allocation failure.

CL_ERR_NOT_EXIST: Logical address is passed and there is no mapping for it in the transparency layer.

CL_IOC_ERR_INVALID_SESSION: Session based communication is requested and the destination is moved to a different location.

CL_IOC_ERR_FLOW_XOFF_STATE: Destination has sent an `XOFF` message.

CL_IOC_ERR_HOST_UNREACHABLE: Host is not reachable.

CL_ERR_BUFFER_OVERRUN: Priority queue has no space left for this message.

CL_ERR_TIMEOUT: Send operation cannot be completed within the specified timeout interval.

CL_ERR_UNSPECIFIED: An unexpected error has occurred.

Description:

This function is used to send a message on the communication port. The message passed can be persistent or non persistent, as specified in the `messageType` field of the `ClIocSendOptionT` structure. Multiple threads can use the same communication port to send a message. If the destination address is a logical address, IOC sends the message to the corresponding physical address. If the destination node address is a broadcast address, the message is sent to all the existing nodes.

Library File:

CIIoc

Related Function(s):

clIocCommPortCreate, clIocReceive, clIocCommPortDelete

3.2.4 clIocReceive

clIocReceive

Synopsis:
Receives message on communication port.

Header File:
clIocApi.h

Syntax:

```
ClRcT clIocReceive(
                  CL_IN ClIocCommPortHandleT commPortHdl,
                  CL_IN ClIocRecvOptionT *pRecvOption,
                  CL_INOUT ClBufferHandleT userMsg,
                  CL_OUT ClIocRecvParamT *pRecvParam );
```

Parameters:

commPortHdl: (in)Handle of the communication port.

pRecvOption :(in)Structure used to pass receive options. If it is NULL, the structure uses the default values.

userMsg: (in/out) Handle to the message that contains the received data. This must be freed by the user after the function returns.

pRecvParam: (out)Parameter related to the priority of the message, origin of the message, length of the message, and protocol. This information is returned when the function executes successfully. This parameter cannot be NULL.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_HANDLE: Communication port handle is invalid.

CL_ERR_NULL_POINTER: `pRecvParam` is NULL.

CL_ERR_INVALID_BUFFER: `userMsg` is invalid.

CL_IOC_ERR_TRY_AGAIN: There is no message present on the non-blocking communication port.

CL_ERR_TIMEOUT: No information received within timeout interval specified in the `pRecvOption`.

CL_IOC_ERR_RECV_UNBLOCKED: Receiver is unblocked.

Description:

This function is used to receive a message on the communication port. The messages are received as per the priority. The behavior of this call depends on the current mode of the port: blocking or non-blocking mode. Multiple threads can be blocked on the same communication port for receiving data.
If the mode is set to blocking and there is no data for the commport, the receiver is blocked. If the mode is non-blocking and there is no data, the error, `CL_IOC_ERR_TRY_AGAIN` is returned. To retrieve the data, the application needs to poll the communication port.

Library File:
ClIoc

Related Function(s):
clIocCommPortCreate, clIocSend, clIocCommPortDelete

3.2.5 clIocTransparencyRegister

clIocTransparencyRegister

Synopsis:
Registers an application with the transparency layer.

Header File:
clIocApi.h

Syntax:
```
ClRcT clIocTransparencyRegister(
                CL_INOUT ClIocTLInfoT*  pTLInfo);
```

Parameters:
pTLInfo: (in/out) This parameter contains the information about the registration. For example, logical address, component ID, context, HA state, and replica semantics.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pTLInfo` is NULL.

CL_IOC_TL_ERR_LIMIT_EXCEEDED: There is no space left in registration database.

CL_ERR_INVALID_PARAMETER: A parameter is not set correctly.

CL_ERR_NO_MEMORY: Memory allocation failure.

CL_IOC_TL_ERR_DUPLICATE_ENTRY: The entry already exists.

Description:
The application uses this function to register its logical address to physical address mapping. It also provides HA State, component ID, context, replica semantics information as part of this mapping. IOC provides location transparency using this information. A given logical address can have multiple physical address mappings.

Library File:
ClIoc

Related Function(s):
clIocTransparencyDeregister, clIocTransparencyLogicalToPhysicalAddrGet

3.2.6 clIocTransparencyDeregister

clIocTransparencyDeregister

Synopsis:
De-registers the application with the transparency layer.

Header File:
clIocApi.h

Syntax:
```
ClRcT clIocTransparencyDeregister(
                        CL_IN ClUint32T compId);
```

Parameters:
compId: (in) ID of the component being de-registered.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is called to de-register the application, identified by component ID, from the transparency layer. The IOC removes the corresponding registered information for this component ID.

Library File:
CIIoc

Related Function(s):
clIocTransparencyRegister

3.2.7 clIocTransparencyLogicalToPhysicalAddrGet

clIocTransparencyLogicalToPhysicalAddrGet

Synopsis:
Retrieves the physical addresses for a given logical address.

Header File:
clIocApi.h

Syntax:

```
ClRcT clIocTransparencyLogicalToPhysicalAddrGet(
     CL_IN    ClIocLogicalAddressT   logicalAddr,
     CL_OUT   ClIocTLMappingT        **pPhysicalAddr,
     CL_OUT   ClUint32T              *pNoEntries);
```

Parameters:

logicalAddr: (in) Logical address of the component.

pPhysicalAddr: (out) List of the physical addresses and its HA state. IOC allocates the memory and it is the responsibility of the application to free it.

pNoEntries: (out) Number of entries in *pPhysicalAddr* list.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_EXIST: Logical address entry is not found in transparency layer.

CL_IOC_TL_ACTIVE_INST_NOT_PRESENT: No entry found for any active instance.

Description:
This function is used to query for the physical addresses. It returns all the physical addresses of the given logical address. The function allocates memory and returns it in `pPhysicalAddr` parameter, and it must be freed by the caller of the function.

Library File:
Cl Ioc

Related Function(s):
clIocTransparencyRegister

3.2.8 clIocLocalAddressGet

clIocLocalAddressGet

Synopsis:
Retrieves the local IOC node address.

Header File:
clIocApi.h

Syntax:

```
ClIocNodeAddressT clIocLocalAddressGet(void);
```

Parameters:
None.

Return values:
This function returns the local IOC node address. If an error occurs, the function returns zero.

Description:
This function returns the IOC node address of the current node.

Library File:
Cliloc

Related Function(s):
None.

3.2.9 clIocTransparencyDeregisterNode

clIocTransparencyDeregisterNode

Synopsis:
De-registers the node with the transparency layer.

Header File:
clIocApi.h

Syntax:

```
ClRcT clIocTransparencyDeregisterNode(
            CL_IN ClIocNodeAddressT nodeId);
```

Parameters:
nodeId (in) ID of the node going down.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is called to de-register the node identified by node ID, from the transparency layer. The IOC removes the corresponding registered information for this node ID.

Library File:
ClIoc

Related Function(s):
clIocTransparencyRegister

3.2.10 clIocBind

clIocBind

Synopsis:
Binds to a particular transport.

Header File:
clIocApi.h

Syntax:

```
 ClRcT clIocBind(
CL_IN ClNameT *toName,
CL_OUT ClIocToBindHandleT *pToHandle);
```

Parameters:

toName (in) Name of the transport to which user wants to bind.

pToHandle (out) Bind handle returned by IOC.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NULL_POINTER: `pToHandle` is NULL.

CL_IOC_ERR_XPORT_LINK_NOT_REGISTERED: There is no transport object or the link is not registered.

Description:
This function takes the transport name as the input and binds IOC to it (it must be a registered transport). The handle returned by this function can be used for sending the data through the corresponding transport. By default, IOC is bound to the highest priority transport, and data can be sent on this transport.

Library File:
Clloc

Related Function(s):
clIocSend

3.2.11 clIocArpInsert

clIocArpInsert

Synopsis:
Adds an ARP entry into the ARP table.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocArpInsert(
          CL_IN ClIocArpParamT *pArpInfo);
```

Parameters:

pArpInfo: (in) The parameters related to the address resolution are passed in this structure.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: If `pArpInfo` is NULL.

CL_ERR_INVALID_PARAMETER: Address is a local address or broadcast address, the address size is greater than `CL_IOC_MAX_XPORT_ADDR_SIZE`, transport is never registered, the status is incorrect, or the entry type is invalid.

CL_IOC_ERR_XPORT_LINK_NOT_REGISTERED: Link is not registered.

CL_ERR_NO_MEMORY: Memory is not available.

CL_ERR_ALREADY_EXIST: A static type entry already exists while adding a dynamic entry.

Description:
This function is used to add an ARP entry into the ARP Table. It is added, if the corresponding link of the current node is registered with IOC.

Library File:
libCIIoc

Related Function(s):
clIocArpDelete

3.2.12 clIocArpDelete

clIocArpDelete

Synopsis:
Deletes an ARP entry from the ARP table.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocArpDelete(
       CL_IN ClIocNodeAddressT iocAddr,
       CL_IN ClUint8T *pXportName,
       CL_IN ClUint8T* pLinkName);
```

Parameters:

iocAddress: (in) Address of the blade whose static ARP entry is deleted.

pXportName: (in) Name of the transport.

pLinkName: (in) Name of the link.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportName` or `pLinkName` is NULL.

CL_ERR_INVALID_PARAMETER: A parameter is not set correctly.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport does not exist.

CL_ERR_NO_MEMORY: Memory allocation failure.

Description:
This function is used to delete an ARP entry from the ARP Table. `pXportName` and `pLinkName` are used to search for a link in the table.

Library File:
libClIoc

Related Function(s):
clIocArpInsert

3.2.13 clIocCommPortWaterMarksGet

clIocCommPortWaterMarksGet

Synopsis:
Retrieves the low and high watermark.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocCommPortWaterMarksGet(
            CL_IN ClUint32T commPort,
            CL_OUT ClUint32T* pLowWaterMark,
            CL_OUT ClUint32T* pHighWaterMark);
```

Parameters:

commPort: (in) Handle of the communication object returned after the communication port is created.

pLowWaterMark (out): Low watermark, in bytes.

pHighWaterMark (out): High watermark, in bytes.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_HANDLE: Communication port handle is invalid.

CL_ERR_NULL_POINTER: `pLowWaterMark` or `pHighWaterMark` is NULL.

Description:
This function returns the low and high watermark for the corresponding communication port.

Library File:
libClIoc

Related Function(s):
clIocCommPortWaterMarksSet

3.2.14 clIocCommPortWaterMarksSet

clIocCommPortWaterMarksSet

Synopsis:
Sets the low and high watermark.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocCommPortWaterMarksSet(
            CL_IN ClUint32T commPort,
            CL_IN ClUint32T lowWaterMark,
            CL_IN ClUint32T highWaterMark);
```

Parameters:

commPort: (in) Handle of the communication object returned after creation of the communication port.

lowWaterMark: (in) Low watermark, in bytes.

highWaterMark: (in) High watermark, in bytes. If *highWaterMark* is 0, the flow control on the given communication port is disabled.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_HANDLE: Communication port handle is invalid.

CL_ERR_INVALID_PARAMETER: Invalid parameter.

CL_ERR_OUT_OF_RANGE: Parameters are out-of-range.

Description:
This function is used to set the low and high watermark of a communication port. This allows the application to set its own toggle levels for sending flow control, `XOFF/XON` messages.

Library File:
libCIIoc

Related Function(s):
clIocCommPortWaterMarksGet

3.2.15 clIocHeartBeatStart

clIocHeartBeatStart

Synopsis:
Starts the heartbeat.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocHeartBeatStart();
```

Parameters:
None.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is used to start the heartbeating process. This is started if all the links support the broadcast. If any of the links does not support broadcast, heartbeat must not be started.

Library File:
libCIIoc

Related Function(s):
clIocHeartBeatStop

3.2.16 clIocHeartBeatStop

clIocHeartBeatStop

Synopsis:
Stops heartbeating.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocHeartBeatStop();
```

Parameters:
None.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is used to stop the heartbeating process. This can be used to stop the IOC from sending the heartbeat. If any of the links does not support the heartbeat, this function can be used to stop the heartbeat process.

Library File:
libCIIoc

Related Function(s):
clIocHeartBeatStart

3.2.17 clIocUdpXportConfigInitialize

clIocUdpXportConfigInitialize

Synopsis:
Initializes and configures UDP transport and link.

Header file:
clIocUdpTransportApi.h

Syntax:
```
ClRcT clIocUdpXportConfigInitialize(
                    CL_IN ClIocUserTransportConfigT *pXportConfig);
```

Parameters:
pXportConfig: (in) Contains the UDP transport and link configuration.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportConfig` is NULL.

Description:
This function is used to configure and initialize the UDP transport and link. The application calls this function when it retrieves all the required configuration information required by UDP.

Library File:
libCIIoc

Related Function(s):
clIocUdpXportFinalize

3.2.18 clIocUdpXportFinalize

clIocUdpXportFinalize

Synopsis:
Frees the UDP transport and link.

Header file:
clIocUdpTransportApi.h

Syntax:

```
ClRcT clIocUdpXportFinalize();
```

Parameters:
None.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC library is not initialized.

Description:
This function is used to free the UDP transport and its links. After this function is successfully executed, no communication is possible through this transport.

Library File:
libCIIoc

Related Function(s):
clIocUdpXportConfigInitialize

3.2.19 clIocRouteInsert

clIocRouteInsert

Synopsis:
Adds a new route entry for the destination.

Header File:
clIocManagementApi.h

Syntax:
```
ClRcT clIocRouteInsert(
       CL_IN ClIocRouteParamT *pRouteInfo);
```

Parameters:
pRouteInfo: (in) Structure for the information related to the route. If it is NULL, an appropriate error message is returned.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pRouteInfo` is NULL.

CL_ERR_INVALID_PARAMETER: The address is a local address or broadcast address, transport is never registered, the flag is not correct, the status is not correct, the entry type is invalid, or the prefix length is out-of-range.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport is not registered.

CL_IOC_ERR_XPORT_LINK_NOT_REGISTERED: Link is not registered.

CL_ERR_NO_MEMORY: Memory is not available.

CL_ERR_ALREADY_EXIST: A dynamic entry is being added when a static type entry already exists.

Description:
This function is used to add a new route entry into the routing table. This is added, if the corresponding link of the current node is registered with IOC.

Library File:
libCIIoc

Related Function(s):
clIocRouteDelete

3.2.20 clIocRouteDelete

clIocRouteDelete

Synopsis:
Deletes the route of the given blade from the routing database.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocRouteDelete(
       CL_IN ClIocNodeAddressT destAddr,
       CL_IN ClUint16T  prefixLen);
```

Parameters:

destAddress: (in) The IOC destination address.

prefixLen: (in) The length of prefix.

Return values:

CL_OK: The function executed successfully .

CL_ERR_INVALID_PARAMETER: Prefix length is incorrect, or `destAddr` is local or broadcast address.

CL_IOC_ERR_ROUTE_NOT_EXIST: There is no matching route.

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is used to delete the route to the given destination address of the blade from the routing database. The blade address and the prefix length are needed to search the route.

Library File:
libClIoc

Related Function(s):
clIocRouteInsert

3.2.21 clIocVersionCheck

clIocVersionCheck

Synopsis:
Checks for the appropriate version of IOC.

Header File:
clIocApi.h

Syntax:
```
ClRcT clIocVersionCheck(
              CL_INOUT ClVersionT *pVersion);
```

Parameters:
pVersion: (in/out) Pointer to the version information.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NULL_POINTER: `pVersion` is NULL.

Description:
This function verifies, if the version specified by the application matches with any of the versions supported by IOC. If it does not match, IOC returns an error and if it matches, IOC populates `pVersion` with the matching version information and this parameter is returned by this function.

Library File:
ClIoc

Related Function(s):
None.

3.2.22 clIocLinkStatusGet

clIocLinkStatusGet

Synopsis:
Returns the status of the link.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocLinkStatusGet(
       CL_IN ClUint8T *pXportName,
       CL_IN ClUint8T* pLinkName,
       CL_OUT ClUint8T *pStatus);
```

Parameters:

pXportName: (in) Name of the transport used by the link.

pLinkName: (in) Name of the link whose status is queried.

pStatus (out): Pointer to the status of the link.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportName` or `pStatus` is NULL.

CL_ERR_INVALID_PARAMETER: Size of `pXportName` or `pLinkName` is greater or equal to `CL_IOC_MAX_XPORT_NAME_LENGTH`.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport is not registered.

CL_IOC_ERR_XPORT_LINK_NOT_REGISTERED: Link is not registered.

Description:
This function returns the status of the link. `pXportName` and `pLinkName` are used to search for the link and transport tables. The status can be one of the following,

1. `CL_IOC_LINK_UP`
2. `CL_IOC_LINK_DOWN`

Library File:
libClIoc

Related Function(s):
clIocLinkStatusSet

3.2.23 clIocLinkStatusSet

clIocLinkStatusSet

Synopsis:
Sets the status of the link.

Header File:
clIocManagementApi.h

Syntax:

```
ClRcT clIocLinkStatusSet(
          CL_IN ClUint8T *pXportName,
          CL_IN ClUint8T* pLinkName,
          CL_IN ClUint8T status);
```

Parameters:

pXportName: (in) Name of the transport used by the link.

pLinkName: (in) Name of the link whose status is to be set.

status: (in) New status of the link. The status can be `CL_IOC_ROUTE_UP` or `CL_IOC_ROUTE_DOWN`.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportName` or `pLinkName` is NULL.

CL_ERR_INVALID_PARAMETER: Size of `pXportName` or `pLinkName` is greater or equal to `CL_IOC_MAX_XPORT_NAME_LENGTH` or status is not valid.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport is not registered.

CL_IOC_ERR_XPORT_LINK_NOT_REGISTERED: Link is not registered.

Description:
This function is used to set the status of the link. `pXportName` and `pLinkName` are used to search for the link and transport tables. The status can be one of the following,

1. `CL_IOC_LINK_UP`
2. `CL_IOC_LINK_DOWN`

Library File:
libClIoc

Related Function(s):
clIocLinkStatusGet

3.2.24 clIocTransportRegister

clIocTransportRegister

Synopsis:
Registers a transport object with the IOC.

Header File:
clIocTransportApi.h

Syntax:

```
ClRcT clIocTransportRegister(
       CL_IN ClIocTransportConfigT *pXportObjConfig);
```

Parameters:

pXportObjConfig: (in) Address of the structure that contains the transport configuration information.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportObjConfig` is NULL.

CL_ERR_VERSION_MISMATCH: IOC library version of the client and server are incompatible.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport registration failure.

CL_IOC_ERR_XPORT_ALREADY_REGISTERED: Transport is already registered.

Description:
This function is used to register a transport object with the IOC. It must be called after the initialization of the underlying device.

Library File:
libClIoc

Related Function(s):
clIocTransportDeregister

3.2.25 clIocTransportDeregister

clIocTransportDeregister

Synopsis:
De-registers the given transport.

Header File:
clIocTransportApi.h

Syntax:

```
ClRcT clIocTransportDeregister(
          CL_IN ClCharT *pXportName);
```

Parameters:
pXportName: (in) Name of the transport. It is a string of length, 128 bytes.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportName` is NULL.

CL_IOC_ERR_XPORT_NOT_REGISTERED: The transport is not registered.

Description:
This function is used to de-register the given transport from the IOC. After the transport is de-registered, no operations related to this transport can be performed.

Library File:
libClIoc

Related Function(s):
clIocTransportRegister

3.2.26 clIocLinkRegister

clIocLinkRegister

Synopsis:
Registers a transport link.

Header File:
clIocTransportApi.h

Syntax:

```
ClRcT clIocLinkRegister(
          CL_IN ClIocTransportLinkConfigT* pXportLinkConfig);
```

Parameters:

pXportLinkConfig: (in) Pointer to a structure that contains the transport link related configuration.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportLinkConfig` is NULL.

CL_ERR_INVALID_PARAMETER: A parameter is not set correctly.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport is not registered.

CL_ERR_NO_MEMORY: Memory allocation failure.

Description:
This function registers the transport link with the IOC. This transport must be registered with the IOC.

Library File:
libCIIoc

Related Function(s):
clIocTransportRegister, clIocLinkDeregister

3.2.27 clIocLinkDeregister

clIocLinkDeregister

Synopsis:
De-registers a transport.

Header File:
clIocTransportApi.h

Syntax:

```
ClRcT clIocLinkDeregister(
       CL_IN ClUint8T *pXportLinkName,
       CL_IN ClUint8T *pXportName);
```

Parameters:

pXportLinkName: (in) Name of the link. It is a string of length, 128 bytes.

pXportName: (in) Name of the transport, with which it is registered.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pXportLinkName` or `pXportName` is NULL.

CL_IOC_ERR_XPORT_NOT_REGISTERED: Transport is not registered.

CL_IOC_ERR_XPORT_LINK_NOT_REGISTERED: Link is not registered.

CL_IOC_ERR_XPORT_LINK_NOT_DELETED: Link cannot be removed.

Description:
This function is used to de-register a transport link from the IOC. This function needs to be called before the transport is de-registered.

Library File:
libCIIoc

Related Function(s):
clIocTransportRegister, clIocTransportDeregister, clIocLinkRegister

3.2.28 clIocCommPortGet

clIocCommPortGet

Synopsis:
Returns the port ID.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocCommPortGet(
                    CL_IN ClIocCommPortHandleT pIocCommPort,
                    CL_OUT ClIocPortT* pPortId);
```

Parameters:

plocCommPort: (in) Handle to the communication port.

pPortId: (out) Pointer to the port ID.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: The IOC is not initialized.

CL_ERR_NULL_POINTER: `pPortId` is NULL.

Description:
This function returns the port ID for a given communication port handle. It needs to be called when the communication port related parameters are required to be set. This function can be called, if the communication port is created through `clIocCommPortCreate()` function.

Related APIs:
clIocCommPortCreate(), clIocSend(), clIocReceive(), clIocCommPortDelete(), clIocLastErrorGet().

3.2.29 clIocCommPortReceiverUnblock

clIocCommPortReceiverUnblock

Synopsis:
Unblocks all receive calls.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:
```
ClRcT clIocCommPortReceiverUnblock(
                    CL_IN ClIocCommPortHandleT commPortHdl);
```

Parameters:
commPortHdl: (in) Handle of the communication port to be unblocked.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_HANDLE: The communication port handle is invalid.

Description:
This function is used to unblock the `receive` calls that are blocked inside IOC on the given communication port. The blocked `receive` calls are unblocked and returns `CL_IOC_RECV_UNBLOCKED`. The `receive` on this communication port stops after this call. To start the `receive` again, the `clIocCommPortBlockRecvSet` function must be called.

Related APIs:
clIocCommPortCreate(), clIocCommPortDelete(), clIocCommPortModeSet(), clIocCommPortModeGet(), clIocCommPortBlockRecvSet(), clIocCommPortDebug().

3.2.30 clIocMaxPayloadSizeGet

clIocMaxPayloadSizeGet

Synopsis:
Returns the maximum size of the payload.

Header File:
clIocApi.h

Library Files:
libCIIoc

Syntax:

```
ClRcT clIocMaxPayloadSizeGet(
                    CL_OUT ClUint32T *pSize);
```

Parameters:
pSize: (out) Contains the maximum size supported for the payload.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pSize` is NULL.

Description:
This function returns the maximum size of the payload that can be sent over the IOC. It does not include the IOC header size.

Note:
In this release, there is no limit over the payload size in IOC. So, this function may not be very useful.

Related APIs:
None.

3.2.31 clIocTotalNeighborEntryGet

clIocTotalNeighborEntryGet

Synopsis:
Returns the total number of neighbor nodes.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocTotalNeighborEntryGet(
                  CL_OUT ClUint32T  *pNumberOfEntries);
```

Parameters:
pNumberOfEntries: (out) Number of neighbor nodes.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pNumberOfEntries` is NULL.

Description:
This function returns the total number of neighbor nodes (including duplicates and local) of the current node. This function should be called before the `clIocNeighborListGet()` function is called.

Related APIs:
clIocNeighborListGet().

3.2.32 clIocNeighborListGet

clIocNeighborListGet

Synopsis:
Returns the list of neighboring IOC nodes.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:
```
ClRcT clIocNeighborListGet(
                CL_INOUT ClUint32T *pNumberOfEntries,
                CL_OUT ClIocNodeAddressT *pAddrList);
```

Parameters:
pNumberOfEntries: (in/out) The number of entries the array can hold. IOC modifies this number, if it fills less number of entries in the `pAddrList` array.

pAddrList: (out) The array of IOC node address.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: Either `pNumberOfEntries` or `pAddrList` is NULL.

CL_ERR_NO_MEMORY: The memory allocation or any other resource allocation has failed.

Description:
This function returns the list of neighboring IOC nodes including the local node. An array of `ClIocNodeAddressT` and the number of entries the array can hold is taken as input. If the number of entries is less than the given length of the array, `pNumberOfEntries` will contain the exact number of entries.
The `clIocTotalNeighborEntryGet()` function should be called to obtain the total number of neighbors and the space to store the addresses can be allocated.

Related APIs:
clIocTotalNeighborEntryGet().

3.2.33 clIocAddressForPhySlotGet

clIocAddressForPhySlotGet

Synopsis:
Returns the IOC address of the node.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocAddressForPhySlotGet(
                    CL_IN ClUint32T phySlotAddr,
                    CL_OUT ClIocNodeAddressT *pIocNodeAddr);
```

Parameters:

phySlotAddr: (in) Physical slot, whose IOC address is queried.

plocNodeAddr: (out) IOC address of the node.

Return values:

CL_OK: The function executed successfully..

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pIocNodeAddr` is NULL.

CL_ERR_NOT_EXIST: The entry does not exist.

Description:
This function returns the IOC address of a node, whose physical slot number is known.

Related APIs:
clIocAddressForPhySlotSet(), clIocPhySlotForIocAddressGet().

3.2.34 clIocAddressForPhySlotSet

clIocAddressForPhySlotSet

Synopsis:
Creates the mapping of IOC address of the node to the physical slot address.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocAddressForPhySlotSet(
                    CL_IN ClUint32T phySlotAddr,
                    CL_IN ClIocNodeAddressT iocNodeAddr);
```

Parameters:
phySlotAddr: (in) Physical slot address of the node .

iocNodeAddr: (in) Address of the IOC node.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function creates the mapping of IOC address of the node to a given physical slot. This call overwrites the previous mapping of IOC node address to slot address.

Related APIs:
clIocAddressForPhySlotGet(), clIocLocalAddressGet(), clIocPhySlotForIocAddressGet().

3.2.35 clIocPhySlotForIocAddressGet

clIocPhySlotForIocAddressGet

Synopsis:
Returns the physical slot address of a given node.

Header File:
clIocApi.h

Library Files:
libCIIoc

Syntax:

```
ClRcT clIocPhySlotForIocAddressGet(
                CL_IN ClIocNodeAddressT iocNodeAddr,
                CL_OUT ClUint32T *pPhySlot );
```

Parameters:

iocNodeAddr: (in) IOC address of the node.

pPhySlot: (out) Physical slot address.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pPhySLot` is NULL

CL_ERR_NOT_EXIST: The entry does not exist.

Description:
This function returns the physical slot of a node, whose IOC node address is passed as the first parameter.

Related APIs:
clIocAddressForPhySlotSet(), clIocAddressForPhySlotGet().

3.2.36 clIocLibInitialize

clIocLibInitialize

Synopsis:
Configures and initializes the IOC.

Header File:
clIocApi.h

Library Files:
libCIIoc

Syntax:

```
ClRcT clIocLibInitialize();
```

Parameters:
None

Return values:
CL_OK: The function executed successfully .

CL_IOC_ERR_INIT_FAILED: IOC initialization has failed.

Description:
This function initializes the IOC and the transport configuration. This function must be called before any other function of IOC can be used.

Related APIs:
clIocLibFinalize(), clIocLibConfigGet().

3.2.37 clIocLibFinalize

clIocLibFinalize

Synopsis:
Frees the IOC.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocLibFinalize();
```

Parameters:
None

Return values:
CL_OK: The function executed successfully .

Description:
This function is used to free the IOC. It de-registers all transport and frees the resources allocated during the initialization of the IOC library.

Related APIs:
clIocLibInitialize().

3.2.38 clIocGeographicalAddressGet

clIocGeographicalAddressGet

Synopsis:
Returns the geographical address of the node.

Header File:
clIocApi.h

Library Files:
libCltoc

Syntax:

```
ClRcT clIocGeographicalAddressGet(
                    CL_IN ClIocNodeAddressT iocNodeAddr,
                    CL_OUT ClCharT *pGeoAddr);
```

Parameters:

iocNodeAddr: (in) Address of the node whose geographical address is queried.

pGeoAddr: (out) Geographical address of the node is returned in this pointer. Memory of size, `CL_MAX_GEO_ADDR_STRING_LENGTH`, must be allocated for this parameter.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_PARAMETER: A parameter is not set correctly.

CL_ERR_NOT_EXIST: The node entry does not exist.

Description:
This function returns the geographical address of the node, whose node address is passed as input.

Related APIs:
clIocGeographicalAddressSet(), clIocLocalAddressGet().

3.2.39 clIocGeographicalAddressSet

clIocGeographicalAddressSet

Synopsis:
Sets the geographical address of the node.

Header File:
clIocApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocGeographicalAddressSet(
                        CL_IN ClIocNodeAddressT iocNodeAddr,
                        CL_IN ClCharT *pGeoAddr);
```

Parameters:

iocNodeAddr: (in) The blade address of the node whose geographical address is to be set.

pGeoAddr: (in) The geographical address of the node. The maximum length of geographical address can be `CL_MAX_GEO_ADDR_STRING_LENGTH`. If you pass NULL in the geographical address, the old entry is removed. If an entry already exists and another string is passed, a new entry will overwrite the old information.

Return values:

CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_INVALID_PARAMETER: `pGeoAddr` is NULL, or `iocNodeAddress` is not a physical address or the length of geographical address is more than `CL_IOC_GEO_ADDR_MAX_LENGTH`.

CL_ERR_UNSPECIFIED: An unexpected error has occurred.

Description:
This function is used to set the geographical address of the node. This function old geographical address with the new address. This can be used to set the geographical address of the local nodes only. The first parameter is from the `clIocLocalAddressGet()` function.

Related APIs:
clIocGeographicalAddressGet(), clIocLocalAddressGet().

3.2.40 clIocRouteTablePrint

clIocRouteTablePrint

Synopsis:
Prints the routing database on the console.

Header File:
clIocManagementApi.h

Library Files:
libCIIoc

Syntax:
```
ClRcT clIocRouteTablePrint();
```

Parameters:
None

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is used to print the routing database on the console. It also checks, if the route to the destination exists.

Related APIs:
clIocRouteInsert(), clIocRouteDelete(), clIocRouteStatusChange(), clIocRoutingTableFlush().

3.2.41 clIocArpTablePrint

clIocArpTablePrint

Synopsis:
Prints an ARP table.

Header File:
clIocManagementApi.h

Syntax:
```
ClRcT clIocArpTablePrint();
```

Parameters:
None

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

Description:
This function is used to print the ARP database on the console. It also checks, if the ARP entry to the destination exists.

Library Files:
libCIIoc

Related APIs:
None.

3.2.42 clIocTransparencyLayerBindingsListShow

clIocTransparencyLayerBindingsListShow

Synopsis:
Prints all entries in a given context.

Header File:
clIocManagementApi.h

Library Files:
libClIoc

Syntax:

```
ClRcT clIocTransparencyLayerBindingsListShow(
                    CL_IN ClUint32T contextId);
```

Parameters:
contextId: (in) ID of the context.

Return values:
CL_OK: The function executed successfully .
CL_ERR_NOT_INITIALIZED: IOC is not initialized.
CL_ERR_INVALID_PARAMETER: `contextId` is invalid.

Description:
This function is used to print all the entries in a given context.

Related APIs:
None.

3.2.43 clIocSessionReset

clIocSessionReset

Synopsis:
Resets the session with a logical address.

Header File:
clIocApi.h

Syntax:
```
ClRcT clIocSessionReset(
                CL_IN ClIocCommPortHandleT iocCommPortHdl,
                CL_IN ClIocLogicalAddressT *pIocLogicalAddress);
```

Parameters:
iocCommPortHdl: (in) Handle to the communication port where the session is maintained.

plocLogicalAddress: (in) Pointer to the logical address for which the session needs to be cleared.

Return values:
CL_OK: The function executed successfully .

CL_ERR_NOT_INITIALIZED: IOC is not initialized.

CL_ERR_NULL_POINTER: `pLogicalAddress` is NULL.

CL_ERR_INVALID_HANDLE: The communication port handle is invalid.

CL_ERR_INVALID_PARAMETER: `pLogicalAddress` is not a logical address.

Description:
This function is used to reset the session with a logical address. If the session is requested with a logical address in a previous `IOCsend` function, that session can be reset using this function.
If the mapping between logical to active physical address changes in a session, all *send* calls return the error, `CL_IOC_ERR_INVALID_SESSION`. This function can be used to restart the session with the new active instance.

Library Files:
libClIoc

Related APIs:
clIocCommPortCreate(), clIocCommPortGet(), clIocLastErrorGet(), clIocCommPortDelete().

Chapter 4

Service Management Information Model

TBD

Chapter 5

Service Notifications

TBD

Chapter 6

Configuration

TBD

Chapter 7

Debug CLIs

TBD

Index