OpenClovis
Software Development Kit (SDK)
Service Description and API Reference for Container Service

For OpenClovis SDK Release2.3 V0.4

Document Revision Date: January 03, 2007

Contents

Chapter 1

Functional Overview

The OpenClovis Container Library provides basic data management facilities by means of Container abstraction. It provides a common interface for all Container types. It provides the following functions:

- Adds a node to the Container.
- Finds a node in a Container when the key is specified.
- Deletes all the nodes associated with a specific key.
- Deletes a specific node from the Container.
- Returns the first node in the Container.
- Returns the last node in the Container.
- Returns the previous node in the Container.
- Returns the next node in the Container.
- Returns the number of nodes associated with a specific key.
- Returns the total number of nodes in the Container.

When adding data into the Container, you must specify a tuple consisting of key and data. The data is associated with specified key. You can associate multiple data with the same key. This can be done by specifying same key when adding data. Before any of the afore-mentioned operations can be performed on the Container, the Container must be created. Container creation is achieved through a specific Container create operation. Currently, the Container Library supports the following three types of specific Containers:

- Doubly Linked list
- Hash-table (supporting open-hashing)
- Red-black trees (balanced binary tree)

It is important to realize that by using the Container abstraction, you can switch Containers without having to rewrite a significant section of their code. Only the Container creation call needs to be changed appropriately. This is explained further with an example in a later section. This document contains the definitions and function prototypes for the Container Library.

Container APIs depend on the Heap utility for memory allocation and deallocation.

Chapter 2

Service APIs

2.1 Type Definitions

2.1.1 ClCntKeyHandleT

typedef ClHandleT ClCntKeyHandleT;

The type of the handle for the user-key.

2.1.2 ClCntKeyCompareCallbackT

*typedef ClInt32T(*ClCntKeyCompareCallbackT)(ClCntKeyHandleT key1, ClCntKeyHandleT key2);*

The type of the callback function registered to compare the keys of a Container.

2.1.3 ClCntDeleteCallbackT

*typedef void(*ClCntDeleteCallbackT)(*
ClCntKeyHandleT userKey,
ClCntDataHandleT userData);

The type of the callback function to delete a node comprising a key and data from the Container.

2.1.4 ClCntKeyTypeT

```
typedef enum {
        CL_CNT_UNIQUE_KEY,
        CL_CNT_NON_UNIQUE_KEY,
        CL_CNT_INVALID_KEY_TYPE
} ClCntKeyTypeT;
```

The *ClCntKeyTypeT* enumeration contains the type of the key for the Container: unique, non-unique or invalid.

2.1.5 ClCntHandleT

typedef ClHandleT ClCntHandleT;

The type of the handle for the Container.

2.1.6 ClCntNodeHandleT

typedef ClHandleT ClCntNodeHandleT;

The type of the handle for the node in a Container.

2.1.7 ClCntDataHandleT

typedef ClHandleT ClCntDataHandleT;

The type of the handle for the user-data.

2.1.8 ClCntWalkCallbackT

*typedef ClRcT(*ClCntWalkCallbackT)(*
ClCntKeyHandleT userKey,
ClCntDataHandleT userData,
ClCntArgHandleT userArg,
ClUint32T dataLength);

The type of the callback function used to traverse/walk through the Container.

2.1.9 ClCntArgHandleT

typedef void ClCntArgHandleT;*

The type of the handle for the user-argument.

2.1.10 ClCntHashCallbackT

*typedef ClUint32T (*ClCntHashCallbackT)(*
ClCntKeyHandleT userKey);

The type of the callback function registered for hash key generation.

2.2 Functional APIs

2.2.1 clCntLlistCreate

clCntLlistCreate

Synopsis:
Creates a Doubly Linked List.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntLlistCreate(
                CL_IN ClCntKeyCompareCallbackT fpKeyCompare,
                CL_IN ClCntDeleteCallbackT fpUserDeleteCallback,
                CL_IN ClCntDeleteCallbackT fpUserDestroyCallback,
                CL_IN ClCntKeyTypeT ContainerKeyType,
                CL_OUT ClCntHandleT* pContainerHandle);
```

Parameters:

fpKeyCompare: (in) Pointer to the user's key compare function. It accepts two parameters of type *ClCntKeyHandleT*. This function returns the following values:

- a negative value: If first key is lesser than the second key.
- zero: If both the keys are equal.
- a positive value: If first key is greater than second key.
  Note:
  For the APIs where a traversal of the nodes in the Container is involved, you must pass a key for a node to be found. This callback function is called by the Container Library for every node in the Container with the key passed by you and the key for that node. The implementation of this function must compare the keys in the application specific way and return a value as documented below.

fpUserDeleteCallback: (in) Pointer to the user's destroy callback function. This function is called by the Container Library whenever either *clCntNodeDelete* or *clCntNodeAllDelete* API is called. The delete callback function must free any memory allocated by the application for the node being deleted.

fpUserDestroyCallback: (in) Pointer to the user's destroy callback function. This function is called, whenever you invoke the destroy API. The user-key and user-data of each node, which is being deleted is passed as first and second argument to this callback.

ClCntKeyTypeT: (in) Enum indicating whether the Container is of a unique key type or a non-unique key type.

pContainerHandle: (out) Pointer to the variable of type ClCntHandleT in which the function returns a valid Container handle on successful creation of Container.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NO_MEMORY: On memory allocation failure.

Description:
This function is used to create a linked list of a Container. It validates whether the function is registered for which the request is made.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeDelete , clCntAllNodesDelete , clCntDelete

2.2.2 clCntHashtblCreate

clCntHashtblCreate

Synopsis:
Creates a Hash Table.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntHashtblCreate(
                  ClUint32T numberOfBuckets,
                   ClCntKeyCompareCallbackT fpKeyCompare,
                   ClCntHashCallbackT fpHashFunction,
                   ClCntDeleteCallbackT fpUserDeleteCallback,
                   ClCntDeleteCallbackT fpUserDestroyCallback,
               ClCntKeyTypeT ContainerKeyType,
                   ClCntHandleT* pContainerHandle);
```

Parameters:

numberOfBuckets: Number of buckets (table size) in the hash table.

fpKeyCompare: (in) Pointer to the user's key compare function. It accepts two parameters of type *ClCntKeyHandleT*. This function returns the following values:

- a negative value: If first key is lesser than the second key.
- zero: If both the keys are equal.
- a positive value: If first key is greater than second key.
  Note:
  For the APIs where a traversal of the nodes in the Container is involved, you must pass a key for a node to be found. This callback function is called by the Container Library for every node in the Container with the key passed by you and the key for that node. The implementation of this function must compare the keys in the application specific way and return a value as documented below.

fpHashFunction: Pointer to the user specified hash function. It accepts one parameter of type ClCntKeyHandleT and returns an integer which is the hash value.

fpUserDeleteCallback: (in) Pointer to the user's destroy callback function. This function is called by the Container Library whenever either *clCntNodeDelete* or *clCntNodeAllDelete* API is called. The delete callback function must free any memory allocated by the application for the node being deleted.

fpUserDestroyCallback: (in) Pointer to the user's destroy callback function. This function is called, whenever you invoke the destroy API. The user-key and user-data of each node, which is being deleted is passed as first and second argument to this callback.

ClCntKeyTypeT: (in) Enum indicating whether the Container is of a unique key type or a non-unique key type.

pContainerHandle: (out) Pointer to the variable of type ClCntHandleT in which the function returns a valid Container handle on successful creation of Container.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NO_MEMORY: On memory allocation failure.

Description:
This function is used to create and initialize Hash Table Containers with the requested table size.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeDelete , clCntAllNodesDelete , clCntDelete

2.2.3 clCntRbtreeCreate

clCntRbtreeCreate

Synopsis:
Creates a Red Black Tree.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntRbtreeCreate(
                ClCntKeyCompareCallbackT fpKeyCompare,
                     ClCntDeleteCallbackT fpUserDeleteCallback,
                     ClCntDeleteCallbackT fpUserDestroyCallback,
                ClCntKeyTypeT ContainerKeyType,
                     ClCntHandleT* pContainerHandle);
```

Parameters:

fpKeyCompare: (in) Pointer to the user's key compare function. It accepts two parameters of type *ClCntKeyHandleT*. This function returns the following values:

- a negatively value: If first key is lesser than the second key.
- zero: If both the keys are equal.
- a positive value: If first key is greater than second key.
  Note:
  For the APIs where a traversal of the nodes in the Container is involved, you must pass a key for a node to be found. This callback function is called by the Container Library for every node in the Container with the key passed by you and the key for that node. The implementation of this function must compare the keys in the application specific way and return a value as documented below.

fpUserDeleteCallback: (in) Pointer to the user's destroy callback function. This function is called by the Container Library whenever either *clCntNodeDelete* or *clCntNodeAllDelete* API is called. The delete callback function must free any memory allocated by the application for the node being deleted.

fpUserDestroyCallback: (in) Pointer to the user's destroy callback function. This function is called, whenever you invoke the destroy API. The user-key and user-data of each node, which is being deleted is passed as first and second argument to this callback.

ClCntKeyTypeT: (in) Enum indicating whether the Container is of a unique key type or a non-unique key type.

pContainerHandle: (out) Pointer to the variable of type ClCntHandleT in which the function returns a valid Container handle on successful creation of Container.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NO_MEMORY: On memory allocation failure.

Description:
This function is used to create and initialize Red Black Tree Container that supports only unique keys.

Library File:
libClCnt

Note:

Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):

clCntNodeDelete , clCntAllNodesDelete , clCntDelete

2.2.4 clCntNodeAddAndNodeGet

clCntNodeAddAndNodeGet

Synopsis:
Adds a new node to Container and returns the node handle.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNodeAddAndNodeGet(
                ClCntHandleT ContainerHandle,
                ClCntKeyHandleT userKey,
                ClCntDataHandleT userData,
                ClRuleExprT* pExp,
                    ClCntNodeHandleT* pNodeHandle);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

userKey: Handle of the user key.

userData: User specified data. Memory allocation for this parameter must be done by you.

pExp: An RBE Expression to associate with this new node.

pNodeHandle: (out) Pointer to the variable of type ClCntNodeHandleT in which handle of the node is added.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

CL_ERR_NO_MEMORY: On memory allocation failure.

CL_ERR_DUPLICATE: If user-key already exists and the Container created supports only unique keys.

Description:
This function is used to create and insert a new node into the Container as well as to return the handle of the newly created node.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeAdd , clCntNodeDelete , clCntAllNodesDelete , clCntDelete

2.2.5 clCntNodeAdd

clCntNodeAdd

Synopsis:
Adds a new node to Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNodeAdd(
                ClCntHandleT ContainerHandle,
                ClCntKeyHandleT userKey,
                ClCntDataHandleT userData,
                ClRuleExprT  *rbeExpression);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

userKey: Handle of the user-key.

userData: User specified data. Memory allocation for *userData* must be done by you.

rbeExpression: An RBE expression to associate with the new node.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

CL_ERR_NO_MEMORY: On memory allocation failure.

CL_ERR_DUPLICATE: If user-key already exists and the Container created supports only unique keys.

Description:
This function is used to create and insert a new node into the Container.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeAdd , clCntNodeFind , clCntNodeDelete , clCntAllNodesDelete

2.2.6 clCntAllNodesDelete

clCntAllNodesDelete

Synopsis:
Deletes all the nodes from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntAllNodesDelete(
                    ClCntHandleT ContainerHandle);
```

Parameters:
ContainerHandle: Handle of Container returned by the create API.

Return values:
CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

CL_ERR_NOT_EXIST: If node does not exist.

Description:
This function is used to delete all the nodes from the Container, but does not affect the user-data. You are required to perform memory allocation and de-allocation for data.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeDelete , clCntAllNodesForKeyDelete , clCntDelete

2.2.7 clCntAllNodesForKeyDelete

clCntAllNodesForKeyDelete

Synopsis:
Deletes all the nodes associated with specific key from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntAllNodesForKeyDelete(
                        ClCntHandleT ContainerHandle,
                        ClCntKeyHandleT userKey);
```

Parameters:
ContainerHandle: Handle of Container returned by the create API.

userKey: User specified key.

Return values:
CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

CL_ERR_NOT_EXIST: If node does not exist.

Description:
This function is used to delete all the nodes associated with the specific user key, but does not affect the user-data. You are required to perform memory allocation and de-allocation for data.

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Library File:
libClCnt

Related Function(s):
clCntAllNodesDelete , clCntDelete

2.2.8 clCntNodeDelete

clCntNodeDelete

Synopsis:
Deletes a specific node from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNodeDelete(
                        ClCntHandleT ContainerHandle,
                        ClCntNodeHandleT nodeHandle);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

nodeHandle: Handle of the node to be deleted.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

CL_ERR_NOT_EXIST: If node does not exist.

Description:
This function is used to delete the specific node from the Container, but it does not affect the user data. You are required to perform memory allocation and de-allocation for data.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeAdd

2.2.9 clCntNodeFind

clCntNodeFind

Synopsis:
Finds a specific node in the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNodeFind(
                        ClCntHandleT ContainerHandle,
                        ClCntKeyHandleT userKey,
                        ClCntNodeHandleT* pNodeHandle);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

userKey: User specified key.

pNodeHandle: (out) Pointer to the variable of type ClCntNodeHandleT in which handle of the node is found.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

CL_ERR_NOT_EXIST: If user-key does not exist.

Description:
This function is used to search for a specific node in the Container. It returns the node associated with the specific key from the Container. If multiple nodes are associated with the same key, the node handle of first node associated with the key is returned.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntFirstNodeGet , clCntNextNodeGet , clCntPreviousNodeGet , clCntLastNodeGet , clCntNodeUserKeyGet , clCntDataForKeyGet , clCntNodeUserDataGet

2.2.10 clCntFirstNodeGet

clCntFirstNodeGet

Synopsis:
Returns the first node from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT  clCntFirstNodeGet(
                            ClCntHandleT ContainerHandle,
                            ClCntNodeHandleT* pNodeHandle);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

pNodeHandle: (out) Pointer to the variable of type ClCntNodeHandleT in which handle of the node is found.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to retrieve the first node from the Container.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntNextNodeGet , clCntPreviousNodeGet , clCntLastNodeGet , clCntNodeUserKeyGet , clCntDataForKeyGet ,

2.2.11 clCntLastNodeGet

clCntLastNodeGet

Synopsis:
Returns the last node from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT  clCntLastNodeGet(
                            ClCntHandleT ContainerHandle,
                            ClCntNodeHandleT  currentNodeHandle,
                            ClCntNodeHandleT* pNextNodeHandle);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

currentNodeHandle: Handle of the current node.

pNextNodeHandle: (out) Pointer to the variable of type ClCntNodeHandleT in which the handle of the next node is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to return the last node from the Container.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntFirstNodeGet , clCntNextNodeGet , clCntPreviousNodeGet , clCntNodeUserKeyGet , clCntDataForKeyGet , clCntNodeUserDataGet

2.2.12 clCntNextNodeGet

clCntNextNodeGet

Synopsis:
Returns the next node from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNextNodeGet(
           ClCntHandleT ContainerHandle,
                        ClCntNodeHandleT  currentNodeHandle,
                        ClCntNodeHandleT* pNextNodeHandle);
```

Parameters:

ContainerHandle: Handle of Container returned by the create API.

currentNodeHandle: Handle of current node.

pNextNodeHandle: (out) Pointer to the variable of type ClCntNodeHandleT in which the handle of the next node is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to return the next node after a specific node in the Container.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntFirstNodeGet , clCntPreviousNodeGet , clCntLastNodeGet clCntNodeUserKeyGet , clCntDataForKeyGet , clCntNodeUserDataGet

2.2.13 clCntPreviousNodeGet

clCntPreviousNodeGet

Synopsis:
Returns the previous node from the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntPreviousNodeGet(
              ClCntHandleT ContainerHandle,
                       ClCntNodeHandleT  currentNodeHandle,
                       ClCntNodeHandleT* pPreviousNodeHandle);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

currentNodeHandle: Handle of the current node.

pPreviousNodeHandle: (out) Pointer to the variable of type ClCntNodeHandleT in which the handle of the previous node is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to return the node before a specific node in the Container.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntFirstNodeGet , clCntNextNodeGet , clCntLastNodeGet clCntPreviousNodeGet , clCntNodeUserKeyGet , clCntDataForKeyGet , clCntNodeUserDataGet

2.2.14 clCntWalk

clCntWalk

Synopsis:
Walks through the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntWalk(
            ClCntHandleT ContainerHandle,
            ClCntWalkCallbackT  fpUserWalkCallback,
            ClCntArgHandleT   userArg,
            ClInt32T    length);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

fpUserWalkCallback: Pointer to the user's walk callback function. It accepts the following arguments:

- ClCntKeyHandleT: Handle to user-key of the node
- ClCntDataHandleT: Handle to user-data of the node. The *userKey* and *userData* of each node are passed as first and second arguments to the callback function. User specified *userArg* is also passed.

userArg: User specified argument, which is passed as third parameter to the user walk callback function. This parameter is also passed to the *clRbeExprEvaluate* API.

length: Length of *userArg*. This parameter is also passed to clRbeExprEvaluate API.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to perform a walk through the Container. For each node encountered during the walk, if the RBE associated with the node evaluates to:

- TRUE or NULL: The user specific callback function is called with the key and data handles present in the encountered node.
- FALSE: The callback function is not called for the current node. The walk function continues with the next node.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
* clCntNextNodeGet

2.2.15 clCntNodeUserKeyGet

clCntNodeUserKeyGet

Synopsis:
Returns the user-key from the node.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNodeUserKeyGet(
                  ClCntHandleT ContainerHandle,
                   ClCntNodeHandleT  nodeHandle,
                   ClCntKeyHandleT* pUserKey);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

nodeHandle: Handle of node from which user-key is to be retrieved.

pUserKey: (out) Pointer to the variable of type ClCntNodeHandleT in which the handle of the user-key is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to retrieve the user-key from a specified node.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntFirstNodeGet , clCntNextNodeGet , clCntLastNodeGet , clCntPreviousNodeGet , clCntNodeUserKeyGet , clCntDataForKeyGet , clCntNodeUserDataGet

2.2.16 clCntDataForKeyGet

clCntDataForKeyGet

Synopsis:
Returns the user-data associated with a specified key.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntDataForKeyGet(
                            ClCntHandleT ContainerHandle,
                            ClCntKeyHandleT   userKey,
                            ClCntDataHandleT  *pUserData);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

userKey: User-key, for which associated data is to be retrieved.

pUserData: (out) Pointer to the variable of type *ClCntNodeHandleT* in which the handle of the user-data associated with a specific key is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

CL_ERR_NOT_EXIST: If user-key does not exist.

CL_ERR_NOT_IMPLEMENTED: If this API is used with non-unique keys.

Description:
This function is used to retrieve the user-data associated with a specified key. It must be used only if the keys are unique. For non-unique keys, the function returns an error.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntFirstNodeGet , clCntNextNodeGet , clCntLastNodeGet , clCntPreviousNodeGet , clCntNodeUserKeyGet , clCntNodeUserDataGet

2.2.17 clCntNodeUserDataGet

clCntNodeUserDataGet

Synopsis:
Returns the user-data from the node.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntNodeUserDataGet(
                  ClCntHandleT ContainerHandle,
                           ClCntNodeHandleT  nodeHandle,
                          ClCntDataHandleT* pUserDataHandle);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

nodeHandle: Handle of the node.

pUserDataHandle: (out) Pointer to the variable of type *ClCntNodeHandleT* in which the handle of the user-data associated with a specific key is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to retrieve the user-data from a specific node.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntNodeFind , clCntFirstNodeGet , clCntNextNodeGet , clCntLastNodeGet, clCntPreviousNodeGet , clCntNodeUserKeyGet , clCntNodeUserDataGet , clCntDataForKeyGet ,

2.2.18 clCntSizeGet

clCntSizeGet

Synopsis:
Returns the size of the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntSizeGet(
                          ClCntHandleT ContainerHandle,
                          ClUint32T* pSize);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

pSize: (out) Pointer to the variable of type *ClCntNodeHandleT* in which size of the Container is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to return the total number of nodes in the Container.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntKeySizeGet

2.2.19 clCntKeySizeGet

clCntKeySizeGet

Synopsis:
Returns the number of nodes associated with a user-key.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntKeySizeGet(
                ClCntHandleT ContainerHandle,
                ClCntKeyHandleT userKey,
                ClUint32T* pSize);
```

Parameters:

ContainerHandle: Handle of the Container returned by the create API.

userKey: Handle of the user-key.

pSize: (out) Pointer to the variable of type *ClCntNodeHandleT* in which the number of nodes associated with the specified user-key is returned.

Return values:

CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle or if the Container is empty.

Description:
This function is used to return the number of nodes associated with a specific user-key.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntSizeGet

2.2.20 clCntDelete

clCntDelete

Synopsis:
Destroys the Container.

Header File:
clCntApi.h

Syntax:

```
ClRcT clCntDelete(
                  ClCntHandleT ContainerHandle);
```

Parameters:
ContainerHandle: Handle of Container returned by the create API.

Return values:
CL_OK: The API executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid Container handle.

Description:
This function is used to delete all the nodes and destroy the Container but it does not affect any user-data.

Library File:
libClCnt

Note:
Returned error is a combination of the component id and error code. Use *CL_GET_ERROR_CODE(RET_CODE)* defined in *clCommonErrors.h* to retrieve the error code.

Related Function(s):
clCntLlistCreate clCntHashtblCreate clCntRbtreeCreate clCntAllNodesDelete clCntAllNodesForKeyDelete

Index