



---

# OpenClovis Software Development Kit (SDK) Service Description and API Reference for Circular List Management Service

For OpenClovis SDK Release 2.3 V0.5  
Document Revision Date: March 08, 2007

---

**Copyright © 2007 OpenClovis Inc.**

**All rights reserved**

This document contains proprietary and confidential information of OpenClovis Inc., and may not be used, modified, copied, reproduced, disclosed or distributed in whole or in part except as authorized by OpenClovis Inc. This document is intended for informational use and planning purposes only. All planned features, specifications, and content are subject to change without notice.

**Third-Party Trademarks**

Sun, Sun Microsystems, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group. Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. CLEI is a trademark of Telcordia Technologies, Inc. Adobe, Acrobat, and Acrobat Reader are registered trademarks of Adobe Systems, Inc. All other trademarks, service marks, product names, or brand names mentioned in this document are the property of their respective owners.

**Government Use**

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 12.212 (Commercial Computer Software-Restricted Rights) and DFAR 227.7202 (Rights in Technical Data and Computer Software), as applicable.

**Note:** This document is not subject of the GPL license, even if you have obtained this document as a part of the GPL-ed version of OpenClovis SDK.

# Contents

<b>1</b>	<b>Functional Overview</b>	<b>1</b>
1.1	Interaction with other components . . . . .	1
<b>2</b>	<b>Service Model</b>	<b>3</b>
<b>3</b>	<b>Service APIs</b>	<b>5</b>
3.1	Type Definitions . . . . .	5
3.1.1	ClClistDropPolicyT . . . . .	5
3.1.2	ClClistDeleteCallbackT . . . . .	5
3.1.3	ClClistT . . . . .	5
3.1.4	ClClistDataT . . . . .	6
3.1.5	ClClistNodeT . . . . .	6
3.1.6	ClClistWalkCallbackT . . . . .	6
3.2	Functional APIs . . . . .	7
3.2.1	clClistCreate . . . . .	7
3.2.2	clClistFirstNodeAdd . . . . .	9
3.2.3	clClistLastNodeAdd . . . . .	10
3.2.4	clClistAfterNodeAdd . . . . .	11
3.2.5	clClistBeforeNodeAdd . . . . .	12
3.2.6	clClistNodeDelete . . . . .	13
3.2.7	clClistFirstNodeGet . . . . .	14
3.2.8	clClistLastNodeGet . . . . .	15
3.2.9	clClistNextNodeGet . . . . .	16
3.2.10	clClistPreviousNodeGet . . . . .	17
3.2.11	clClistWalk . . . . .	18
3.2.12	clClistDataGet . . . . .	19
3.2.13	clClistSizeGet . . . . .	20
3.2.14	clClistDelete . . . . .	21

---

## CONTENTS

<b>4</b>	<b>Service Management Information Model</b>	<b>23</b>
<b>5</b>	<b>Service Notifications</b>	<b>25</b>
<b>6</b>	<b>Debug CLIs</b>	<b>27</b>

# Chapter 1

## Functional Overview

A Circular list is a chain of nodes, without any termination. The last node is linked back to the first node. A node is a data structure which contains user data. The following are the operations supported by Circular List implementation:

- Adds a node at the beginning of the list.
- Adds a node at the end of the list.
- Adds a node before a specific node in the list.
- Adds a node after a specific node in the list.
- Returns the first node from the list.
- Returns the last node from the list.
- Returns the previous node from the list.
- Returns the next node from the list.
- Walks through the list, starting from a specific node.
- Deletes a node from the list.
- Returns the number of nodes from the list.
- Retrieves the data from a node from the list.
- Destroys the list.

Before performing any of the above mentioned operations, you must create a list. While creating a list, you need to specify the maximum size for the list. If the maximum size is specified as 0, then you can add any number of nodes. Otherwise, the number of nodes you can add is limited to the maximum size. At any instant, the list can have a maximum of `maxSize` number of nodes, specified when the list is created.

This file contains the definitions and API prototypes for Circular Linked List.

### 1.1 Interaction with other components

Circular List APIs depend on Heap for memory allocation and functions to free the allocated memory.



## **Chapter 2**

# **Service Model**

TBD





## Chapter 3

# Service APIs

### 3.1 Type Definitions

#### 3.1.1 **CIClistDropPolicyT**

```
typedef enum {  
    CL_NO_DROP,  
    CL_DROP_FIRST,  
    CL_DROP_MAX_TYPE  
} CIClistDropPolicyT;
```

The values of the *CIClistDropPolicyT* enumeration contains/decides the action to be taken when the linked list is full. The values of this enumeration have the following interpretation:

- *CL\_NO\_DROP* - The first node must not be dropped even if the list is full.
- *CL\_DROP\_FIRST* - Drops the first node, if the list is full.
- *CL\_DROP\_MAX\_TYPE* - Adds a new drop policy type before the first node.

#### 3.1.2 **CIClistDeleteCallbackT**

```
typedef void(*CIClistDeleteCallbackT)(  
    CIClistDataT userData);
```

The type of the callback function that is invoked when a node is deleted from the container. It deletes the callback function pointer.

#### 3.1.3 **CIClistT**

```
typedef CIHandleT CIClistT;
```

The type of the handle for the circular list.

### **3.1.4 CICollectionDataT**

```
typedef CICollectionHandleT CICollectionDataT;
```

The type of the handle for the user-data.

### **3.1.5 CICollectionNodeT**

```
typedef CICollectionHandleT CICollectionNodeT;
```

The type of the handle for the circular node.

### **3.1.6 CICollectionWalkCallbackT**

```
typedef void(*CICollectionWalkCallbackT)(  
CICollectionDataT userData,  
void *userArg);
```

The type of the callback function to be used when iterating/walking through the elements of a container.

## 3.2 Functional APIs

### 3.2.1 clClistCreate

#### clClistCreate

##### Synopsis:

Creates a Circular Linked list.

##### Header File:

clClistApi.h

##### Syntax:

```
ClRcT clClistCreate(  
    CL_IN  ClUInt32T maxSize,  
    CL_IN  ClClistDropPolicyT dropPolicy,  
    CL_IN  ClClistDeleteCallbackT fpUserDeleteCallBack,  
    CL_IN  ClClistDeleteCallbackT fpUserDestroyCallBack,  
    CL_OUT ClClistT* pListHead);
```

##### Parameters:

**maxSize:** (in) Maximum size of the list. It specifies the maximum number of nodes that can exist at any instant in the list. This must be an unsigned integer. You can add nodes to the list until this maximum limit is reached. If the value of this parameter is set to 0, there is no limit on the size of the list.

**dropPolicy:** (in) Drop policy for the list. If you try to add a node into a list that has reached its maximum limit, and if the drop policy is set to `CL_DROP_FIRST`, the first node is dropped and the specified node is added. If this parameter is set to `CL_NO_DROP`, an error is returned. This parameter is valid, only if the list is of fixed size.

**fpUserDeleteCallBack:** (in) Pointer to the delete callback function of the user. This function accepts a parameter of type `ClClistDataT`. After deleting the specified node, the user-data stored in that node is passed as an argument to the callback function.

**fpUserDestroyCallBack:** (in) Pointer to the destroy callback function. This function accepts a parameter of type `ClClistDataT`. When destroying, the user-data stored in each node is passed as an argument to the callback function, one by one.

**pListHead:** (out) Pointer to the variable of type `ClClistT` in which the function returns a valid list handle on successful creation of the list.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** `pListHead` contains a NULL pointer.

**CL\_ERR\_NO\_MEMORY:** Memory allocation failure.

##### Note:

Returned error is a combination of the component ID and error code. Use `CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

##### Description:

This function creates and initializes the Circular Linked list.

**Library File:**

libCIUtils

**Related Function(s):**

[clClistDelete](#)

## 3.2 Functional APIs

---

### 3.2.2 clClistFirstNodeAdd

#### clClistFirstNodeAdd

**Synopsis:**

Adds a node at the beginning of the list.

**Header File:**

clClistApi.h

**Syntax:**

```
ClRcT clClistFirstNodeAdd(  
    CL_IN ClClistT listHead,  
    CL_IN ClClistDataT userData);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the `clClistCreate` API.

**userData:** (in) User-data. You must allocate and de-allocate memory for this parameter.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NO\_MEMORY:** Memory allocation failure.

**CL\_ERR\_MAXSIZE\_REACHED:** The maximum size is reached.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**Note:**

Returned error is a combination of the component ID and error code. Use

`CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

**Description:**

This function adds a node at the beginning of the Circular Linked list.

**Library File:**

libCIUtils

**Related Function(s):**

[clClistLastNodeAdd](#), [clClistAfterNodeAdd](#), [clClistBeforeNodeAdd](#)

### 3.2.3 `clClistLastNodeAdd`

#### `clClistLastNodeAdd`

**Synopsis:**

Adds a node at the end of the list.

**Header File:**

`clClistApi.h`

**Syntax:**

```
ClRcT clClistLastNodeAdd(  
    CL_IN ClClistT listHead,  
    CL_IN ClClistDataT userData);
```

**Parameters:**

***listHead:*** (in) Handle of the list returned by the create API.

***userData:*** (in) User-data. You must allocate and de-allocate memory for this parameter.

**Return values:**

***CL\_OK:*** The function executed successfully.

***CL\_ERR\_NO\_MEMORY:*** Memory allocation failure.

***CL\_ERR\_MAXSIZE\_REACHED:*** The maximum size is reached.

***CL\_ERR\_INVALID\_HANDLE:*** An invalid handle has been passed to the function.

**Note:**

Returned error is a combination of the component ID and error code. Use

`CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

**Description:**

This function adds a node at the end of the Circular Linked list.

**Library File:**

`libClUtils`

**Related Function(s):**

[clClistAfterNodeAdd](#), [clClistBeforeNodeAdd](#)

## 3.2 Functional APIs

---

### 3.2.4 cClcstAfterNodeAdd

#### cClcstAfterNodeAdd

**Synopsis:**

Adds a node after a specified node in the list.

**Header File:**

cClcstApi.h

**Syntax:**

```
ClRcT cClcstAfterNodeAdd(  
    CL_IN ClClistT listHead,  
    CL_IN ClClistNodeT currentNode,  
    CL_IN ClClistDataT userData);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the create API.

**currentNode:** (in) Handle of the node after which the specified data must be added.

**userData:** (in) User-data. You must allocate and de-allocate memory for this parameter.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NO\_MEMORY:** Memory allocation failure.

**CL\_ERR\_MAXSIZE\_REACHED:** The maximum size is reached.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**Note:**

Returned error is a combination of the component ID and error code. Use

`CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

**Description:**

This function adds a node after a specified node in the Circular Linked list.

**Library File:**

libClUtils

**Related Function(s):**

[cClcstLastNodeAdd](#), [cClcstBeforeNodeAdd](#)

### 3.2.5 cClIstBeforeNodeAdd

#### cClIstBeforeNodeAdd

**Synopsis:**

Adds a node before a specified node in the list.

**Header File:**

cClIstApi.h

**Syntax:**

```
ClRcT  cClIstBeforeNodeAdd(  
                                CL_IN  ClClistT listHead,  
                                CL_IN  ClClistNodeT currentNode,  
                                CL_IN  ClClistDataT userData);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the create API.

**currentNode:** (in) Handle of the node before which the specified data must be added.

**userData:** (in) User-data. You must perform memory allocation and de-allocation for this parameter.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NO\_MEMORY:** Memory allocation failure.

**CL\_ERR\_MAXSIZE\_REACHED:** The maximum size is reached.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**Note:**

Returned error is a combination of the component ID and error code. Use

`CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

**Description:**

This function adds a node before a specified node in the Circular Linked list.

**Library File:**

libClUtils

**Related Function(s):**

[cClIstLastNodeAdd](#), [cClIstAfterNodeAdd](#)



## 3.2 Functional APIs

---

### 3.2.6 clClistNodeDelete

#### clClistNodeDelete

##### Synopsis:

Deletes a node from the list.

##### Header File:

clClistApi.h

##### Syntax:

```
CL_RET clClistNodeDelete(  
    CL_IN ClClistT listHead,  
    CL_IN ClClistNodeT node);
```

##### Parameters:

**listHead:** (in) Handle of the list returned by the create API.

**node:** (in) Handle of the node which is to be deleted.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

##### Note:

Returned error is a combination of the component ID and error code. Use `CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

##### Description:

This function deletes the specified node from the Circular Linked list. The delete callback function, registered during the creation is called with the data in the node to be deleted.

##### Library File:

libCIUtils

##### Related Function(s):

[clClistFirstNodeAdd](#), [clClistLastNodeAdd](#), [clClistAfterNodeAdd](#), [clClistBeforeNodeAdd](#)

### 3.2.7 clClistFirstNodeGet

#### clClistFirstNodeGet

**Synopsis:**

Returns the first node from the list.

**Header File:**

clClistApi.h

**Syntax:**

```
ClRcT clClistFirstNodeGet(  
    CL_IN  ClClistT listHead,  
    CL_OUT ClClistNodeT* pFirstNode);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the create API.

**pFirstNode:** (out) Pointer to the variable of type ClClistNodeT.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pFirstNode contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**CL\_ERR\_NOT\_EXIST:** The list is empty.

**Note:**

Returned error is a combination of the component ID and error code. Use

CL\_GET\_ERROR\_CODE (RET\_CODE) defined in the clCommonErrors.h file to get the error code.

**Description:**

This function returns the first node from the Circular Linked list.

**Library File:**

libCIUtils

**Related Function(s):**

[clClistLastNodeGet](#), [clClistNextNodeGet](#), [clClistPreviousNodeGet](#)

## 3.2 Functional APIs

---

### 3.2.8 clClistLastNodeGet

#### clClistLastNodeGet

##### Synopsis:

Returns the last node from the list.

##### Header File:

clClistApi.h

##### Syntax:

```
ClRcT clClistLastNodeGet (
    CL_IN  ClClistT listHead,
    CL_OUT ClClistNodeT* pLastNode);
```

##### Parameters:

**listHead:** (in) Handle of the list returned by the create API.

**pLastNode:** (out) Pointer to the variable of type ClClistNodeT.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pLastNode contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**CL\_ERR\_NOT\_EXIST:** The list is empty.

##### Note:

Returned error is a combination of the component ID and error code. Use

CL\_GET\_ERROR\_CODE (RET\_CODE) defined in the clCommonErrors.h file to get the error code.

##### Description:

This function is used to return the last node from the Circular Linked list.

##### Library File:

libCIUtils

##### Related Function(s):

[clClistNextNodeGet](#), [clClistPreviousNodeGet](#)

### 3.2.9 clClistNextNodeGet

#### clClistNextNodeGet

**Synopsis:**

Returns the next node from the list.

**Header File:**

clClistApi.h

**Syntax:**

```
ClRcT clClistNextNodeGet (
    CL_IN  ClClistT listHead,
    CL_IN  ClClistNodeT currentNode,
    CL_OUT ClClistNodeT* pNextNode);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the create API.

**currentNode:** (in) Handle of the current node.

**pNextNode:** (out) Pointer to the variable of type ClClistNodeT.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pNextNode contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**CL\_ERR\_NOT\_EXIST:** The list is empty.

**Note:**

Returned error is a combination of the component ID and error code. Use

CL\_GET\_ERROR\_CODE (RET\_CODE) defined in the clCommonErrors.h file to get the error code.

**Description:**

This function is used to return the next node of the specified node from the Circular Linked list.

**Library File:**

libCIUtils

**Related Function(s):**

[clClistLastNodeGet](#), [clClistPreviousNodeGet](#)

## 3.2 Functional APIs

---

### 3.2.10 clClistPreviousNodeGet

#### clClistPreviousNodeGet

##### Synopsis:

Returns the previous node from the list.

##### Header File:

clClistApi.h

##### Syntax:

```
ClRcT clClistPreviousNodeGet (
    CL_IN  ClClistT listHead,
    CL_IN  ClClistNodeT currentNode,
    CL_OUT ClClistNodeT* pPreviousNode);
```

##### Parameters:

**listHead:** (in) Handle of the list returned by the create API.

**currentNode:** (in) Handle of the current node.

**pPreviousNode:** (out) Pointer to the variable of type ClClistNodeT.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pPreviousNode contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**CL\_ERR\_NOT\_EXIST:** The list is empty.

##### Note:

Returned error is a combination of the component ID and error code. Use

CL\_GET\_ERROR\_CODE (RET\_CODE) defined in the clCommonErrors.h file to get the error code.

##### Description:

This function is used to return the previous node of the specified node from the Circular Linked list.

##### Library File:

libCIUtils

##### Related Function(s):

[clClistLastNodeGet](#), [clClistNextNodeGet](#)

### 3.2.11 clClistWalk

#### clClistWalk

**Synopsis:**

Walks through the list.

**Header File:**

clClistApi.h

**Syntax:**

```
ClRcT clClistWalk(  
    CL_IN  ClClistT listHead,  
    CL_IN  ClClistWalkCallbackT fpUserWalkCallBack,  
    CL_IN  void* userArg);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the create API.

**fpUserWalkCallBack:** (in) Pointer to the user callback function. It can have two values:

- The first parameter must be of type `ClClistDataT`.
- The second parameter must be of type `void *`.  
The user-data stored in each node is passed one-by-one as the first argument to the callback function.

**userArg:** (in) User-specified argument. This variable is passed as the second argument to the callback function.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** `userArg` contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**Note:**

Returned error is a combination of the component ID and error code. Use `CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

**Description:**

This function performs a walk through on the Circular Linked list. The user-specified callback function is called with every node's data.

**Library File:**

libClUtils

**Related Function(s):**

None.

## 3.2 Functional APIs

---

### 3.2.12 clClistDataGet

#### clClistDataGet

##### Synopsis:

Retrieves data from a node in the list.

##### Header File:

clClistApi.h

##### Syntax:

```
ClRcT clClistDataGet(  
    CL_IN  ClClistT listHead,  
    CL_IN  ClClistNodeT node,  
    CL_OUT ClClistDataT* pUserData);
```

##### Parameters:

**listHead:** (in) Handle of the list returned by the create API.

**node:** (in) Handle of the node.

**pUserData:** (out) Pointer to the variable of type ClClistDataT.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pUserData contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

##### Note:

Returned error is a combination of the component ID and error code. Use

CL\_GET\_ERROR\_CODE (RET\_CODE) defined in the clCommonErrors.h file to get the error code.

##### Description:

This function retrieves data from the specified node in the list.

##### Library File:

libCIUtils

##### Related Function(s):

None.

### 3.2.13 clClistSizeGet

#### clClistSizeGet

**Synopsis:**

Returns the number of data elements (nodes) in the list.

**Header File:**

clClistApi.h

**Syntax:**

```
ClRcT clClistSizeGet(  
    CL_IN  ClClistT listHead,  
    CL_OUT ClUInt32T* pSize);
```

**Parameters:**

**listHead:** (in) Handle of the list returned by the create API.

**pSize:** (out) Pointer to the variable of type *ClUInt32T*.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** *pSize* contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

**Note:**

Returned error is a combination of the component ID and error code. Use `CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

**Description:**

This function is used to return the number of data elements (nodes) in the list.

**Library File:**

libClUtils

**Related Function(s):**

None.



## 3.2 Functional APIs

---

### 3.2.14 clClistDelete

#### clClistDelete

##### Synopsis:

Destroys the list.

##### Header File:

clClistApi.h

##### Syntax:

```
ClRcT clClistDelete(  
    CL_IN ClClistT* pListHead);
```

##### Parameters:

**pListHead:** (in) Handle of the list returned by the create API.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pListHead contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** An invalid handle has been passed to the function.

##### Note:

Returned error is a combination of the component ID and error code. Use `CL_GET_ERROR_CODE (RET_CODE)` defined in the `clCommonErrors.h` file to get the error code.

##### Description:

This function deletes all the nodes in the list. The destroy callback function, registered during creation, is called for every node in the list with the corresponding data.

##### Library File:

libCIUtils

##### Related Function(s):

[clClistCreate](#)



## **Chapter 4**

# **Service Management Information Model**

TBD



## **Chapter 5**

# **Service Notifications**

TBD



## **Chapter 6**

# **Debug CLIs**

TBD

# Index

clClistAfterNodeAdd, [11](#)  
clClistBeforeNodeAdd, [12](#)  
clClistCreate, [7](#)  
clClistDataGet, [19](#)  
ClClistDataT, [6](#)  
clClistDelete, [21](#)  
ClClistDeleteCallbackT, [5](#)  
ClClistDropPolicyT, [5](#)  
clClistFirstNodeAdd, [9](#)  
clClistFirstNodeGet, [14](#)  
clClistLastNodeAdd, [10](#)  
clClistLastNodeGet, [15](#)  
clClistNextNodeGet, [16](#)  
clClistNodeDelete, [13](#)  
ClClistNodeT, [6](#)  
clClistPreviousNodeGet, [17](#)  
clClistSizeGet, [20](#)  
ClClistT, [5](#)  
clClistWalk, [18](#)  
ClClistWalkCallbackT, [6](#)