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

For OpenClovis SDK Release2.3 FCS2

Document Version: V0.6

Document Revision Date: December 19, 2006

Contents

Chapter 1

Functional Overview

The OpenClovis Buffer Manager Library is designed to provide an efficient method for management of user space buffer and memory in order to increase the performance of communication intensive OpenClovis ASP components and user applications. It provides buffers that can expand or shrink dynamically based on application memory requirement.

The Buffer Manager functions enable you to perform various functions such as, creating or deleting a message, and reading, writing and trimming the number of bytes from a message.

The Buffer Management library also provides the facility of markers. You can set a marker at a specific offset in the message and can restore the write pointer to that offset in the message.

The Buffer Management interacts with the Heap library for allocation and de-allocation of memory.

Chapter 2

Service APIs

2.1 Type Definitions

2.1.1 ClBufferHandleT

typedef ClHandleT ClBufferHandleT;

The type of the handle for the buffer message.

2.1.2 ClBufferSeekTypeT

```
typedef enum {
        CL_BUFFER_SEEK_SET,
        CL_BUFFER_SEEK_CUR,
        CL_BUFFER_SEEK_END,
        CL_BUFFER_SEEK_MAX,
} ClBufferSeekTypeT;
```

The *ClBufferSeekTypeT* enumeration contains the seek types for buffer messages.

2.1.3 ClBufferPoolConfigT

```
typedef struct BufferPoolConfigT {
        ClUint32T numPools;
        ClPoolConfigT *pPoolConfig;
        ClBoolT lazy;
} ClBufferPoolConfigT;
```

The structure *ClBufferPoolConfigT* contains the buffer configuration information.

2.2 Library Life Cycle

2.2.1 clBufferInitialize

clBufferInitialize

Synopsis:
Initializes the Buffer Management library.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferInitialize (const ClPtrT pConfig);
```

Parameters:
pConfig:

Return values:
CL_OK: The function executed successfully.

CL_ERR_NO_MEMORY: On memory allocation failure.

CL_ERR_INITIALIZED: If the Buffer Management library is already initialized.

Description:
This function is used to initialize the Buffer Management library. It must be called before calling any other buffer management function.

Library File:
ClBuffer

Related Function(s):
clBufferFinalize

2.2.2 clBufferFinalize

clBufferFinalize

Synopsis:
Cleans up the Buffer Management library.

Header File:
clBufferApi.h

Syntax:
```
ClRcT clBufferFinalize (void);
```

Parameters:
None

Return values:
CL_OK: The function executed successfully.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is required to be called after you have completed using the buffer management library. It is called during shutdown of components which use the Buffer Management library.

Library File:
ClBuffer

Related Function(s):
clBufferInitialize

2.3 Functional APIs

2.3.1 clBufferCreate

clBufferCreate

Synopsis:
Creates a new message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferCreate(
                            CL_OUT ClBufferHandleT *pMessageHandle);
```

Parameters:
pMessageHandle: (out) A pointer to the handle of type *ClBufferHandleT* designating the message buffer created.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_NO_MEMORY: On memory allocation failure.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to create a new message and pre-allocates a buffer of two kilobytes. It pre-allocates buffers even before a write operation is performed. As you write into the message, the buffer management library will use the pre-allocated buffers. When there are no more buffers to write, the buffer management library will allocate the buffers.

Library File:
ClBuffer

Related Function(s):
clBufferCreateAndAllocate, clBufferDelete, clBufferClear

2.3.2 clBufferCreateAndAllocate

clBufferCreateAndAllocate

Synopsis:
Creates a new message and pre-allocates the buffers.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferCreateAndAllocate(
                                CL_IN ClUint32T size,
                                CL_OUT ClBufferHandleT *pMessageHandle);
```

Parameters:

size: (in) Initial size of message buffer in bytes.

pMessageHandle: (out) A pointer to the handle of type *ClBufferHandleT* designating the message buffer created.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_NO_MEMORY: On memory allocation failure.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to create a new message and pre-allocate buffers. It pre-allocates buffers even before a write operation is performed. As you write into the message, the buffer management library will use the pre-allocated buffers. When there are no more buffers to write, the buffer management library will allocate the buffers.

Library File:
ClBuffer

Related Function(s):
clBufferDelete, clBufferClear , clBufferCreate

2.3.3 clBufferDelete

clBufferDelete

Synopsis:
Deletes a message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferDelete (
                          CL_IN ClBufferHandleT *pMessageHandle);
```

Parameters:
pMessageHandle: (in) Pointer to the message handle returned by the *clBufferCreate()* function.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to delete message buffer(s) designated by the handle obtained through *clBufferCreate()* function. All the markers associated with the message are deleted automatically along with the message buffer(s). Successful invocation of this function will make the handle invalid.

Library File:
ClBuffer

Related Function(s):
clBufferClear , clBufferCreateAndAllocate, clBufferCreate

2.3.4 clBufferClear

clBufferClear

Synopsis:
Deletes the content of a message.

Header File:
clBufferApi.h

Syntax:
```
ClRcT clBufferClear(
                            CL_IN ClBufferHandleT *pMessageHandle);
```

Parameters:
pMessageHandle: (in) Pointer to the message handle returned by *clBufferCreate()* function.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to delete message buffer(s) designated by the handle obtained through *clBufferCreate()* function. All the markers associated with the message are deleted automatically along with the message buffer(s). The handle is still available for re-use.

Library File:
ClBuffer

Related Function(s):
clBufferDelete, clBufferCreateAndAllocate, clBufferCreate

2.3.5 clBufferLengthGet

clBufferLengthGet

Synopsis:
Returns the length of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferLengthGet(
                                CL_IN ClBufferHandleT messageHandle,
                                CL_OUT ClUint32T *pMessageLength);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function.

pMessageLength: (out) Pointer to a variable of type *ClUint32T*, in which the total length of the message is returned.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to retrieve the total length of the message. There is no impact on the read and write offset of the message even on subsequent invocation of this function.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferNBytesWrite

2.3.6 clBufferNBytesRead

clBufferNBytesRead

Synopsis:
Reads the specified number of bytes of data from a message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferNBytesRead(
                          CL_IN ClBufferHandleT messageHandle,
                          CL_OUT ClUint8T *pByteBuffer,
                          CL_INOUT ClUint32T* pNumberOfBytesToRead);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()*.

pByteBuffer: (out) Pointer to a buffer of type *ClUint8T*, in which *numberOfBytesToRead* bytes of data is returned. Memory allocation/deallocation for this parameter must be done by the caller.

pNumberOfBytesToRead: (in/out) Pointer to variable of type *ClUint32T* which contains number of bytes of data to be read. The number of bytes of data that is successfully read is returned in this parameter.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to read *pNumberOfBytesToRead* bytes of data from the message, starting from the current read offset. The current read offset is automatically updated by *pNumberOfBytesToRead* bytes.

Library File:
ClBuffer

Related Function(s):
clBufferClear , clBufferNBytesWrite , clBufferCreate

2.3.7 clBufferNBytesWrite

clBufferNBytesWrite

Synopsis:
Writes the specified number bytes of data to a message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferNBytesWrite(
                          CL_IN ClBufferHandleT messageHandle,
                          CL_IN ClUint8T *pByteBuffer,
                          CL_IN ClUint32T numberOfBytesToWrite);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function into which n bytes of data is to be written.

pByteBuffer: (in) Pointer to the byte buffer from which *numberOfBytesToWrite* bytes are to be written into the message.

numberOfBytesToWrite: (in) Number of bytes to be written.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to write n bytes of data into the message referred by the message handle, starting from the current write offset. The write offset is automatically updated by *numberOfBytesToWrite* bytes.

Library File:
ClBuffer

Related Function(s):
clBufferClear , clBufferNBytesRead , clBufferCreate

2.3.8 clBufferChecksum16Compute

clBufferChecksum16Compute

Synopsis:
Computes a 16-bit checksum on a message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferChecksum16Compute(
            CL_IN ClBufferHandleT messageHandle,
            CL_IN ClUint32T startOffset,
            CL_IN ClUint32T length,
            CL_OUT ClUint16T* pChecksum);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function for which the checksum is to be computed.

startOffset: (in) Offset from the beginning of the message, from which the data for computing the checksum begins.

length: (in) Number of bytes of data for which the checksum is to be computed.

pChecksum: (out) Pointer to variable of type *ClUint16T* in which the computed checksum is returned.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to compute a 16-bit checksum on a message. This checksum can be used to check the validity of the message handle after passing through a network.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferChecksum32Compute

2.3.9 clBufferChecksum32Compute

clBufferChecksum32Compute

Synopsis:
Computes a 32-bit checksum on a message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferChecksum32Compute(
            CL_IN ClBufferHandleT messageHandle,
            CL_IN ClUint32T startOffset,
            CL_IN ClUint32T length,
            CL_OUT ClUint32T* pChecksum);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function for which the checksum is to be computed.

startOffset: (in) Offset from the beginning of the message, from which the data for computing the checksum begins.

length: (in) Number of bytes of data for which the checksum is to be computed.

pChecksum: (out) Pointer to variable of type *ClUint32T* in which the computed checksum is returned.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to compute a 32-bit checksum on a message. This checksum can be used to check the validity of the message handle after passing through a network.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferChecksum16Compute

2.3.10 clBufferDataPrepend

clBufferDataPrepend

Synopsis:
Adds the specified number of bytes at the beginning of message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferDataPrepend(
                        CL_IN ClBufferHandleT messageHandle,
                        CL_IN ClUint8T *pByteBuffer,
                        CL_IN ClUint32T numberOfBytesToWrite);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function.

pByteBuffer: (in) Pointer to a variable of type *ClUint8T*, in which *numberOfBytesToRead* bytes of data is returned. Memory allocation and de-allocation must be done by you.

numberOfBytesToWrite: (in) Number of bytes to be written.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to prepend the specified number of bytes at the beginning of the message. The write offset will be updated and will be equal to *numberOfBytesToWrite*. That is, the next write will begin at *numberOfBytesToWrite* bytes from the starting of the message. Note that the current write offset is lost. Hence, it is advisable to preserve the write offset before invoking this function.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferDataAppend , clBufferDelete, clBufferConcatenate

2.3.11 clBufferConcatenate

clBufferConcatenate

Synopsis:
Concatenates source message to destination message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferConcatenate(
                        CL_IN ClBufferHandleT destination,
                        CL_INOUT ClBufferHandleT *pSource);
```

Parameters:

destination: (in) MessageHandle of destination message. The handle of the concatenated message is returned in this parameter.

pSource: (in/out) Pointer to *messageHandle* of source message, which is to be concatenated with destination message. This handle will be invalid and the source message will no longer exist after this function is called. All the markers in this message will be deleted.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to concatenate the source message to the destination message. All the markers in the source message will be deleted. The write offset of the destination message will be set to end of the message. The read offset of the destination message will be set to start of the message.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferDataAppend , clBufferDelete, clBufferDataPrepend

2.3.12 clBufferReadOffsetGet

clBufferReadOffsetGet

Synopsis:
Returns current read offset of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferReadOffsetGet(
                    CL_IN ClBufferHandleT messageHandle,
                    CL_OUT ClUint32T *pReadOffset);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function.

pReadOffset: (out) Pointer to a variable of type *ClUint32T*, in which the current read offset is returned.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to retrieve the current read offset of the message. This offset is updated on invoking the *clBufferNBytesRead()* function or directly through *clBufferReadOffsetSet()*.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferWriteOffsetGet , clBufferDelete, clBufferReadOffsetSet , clBufferWriteOffsetSet, clBufferTrailerTrim , clBufferHeaderTrim, clBufferNBytesRead

2.3.13 clBufferWriteOffsetGet

clBufferWriteOffsetGet

Synopsis:
Returns current write offset of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferWriteOffsetGet(
                    CL_IN ClBufferHandleT messageHandle,
                    CL_OUT ClUint32T *pWriteOffset);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate* function

pWriteOffset: (out) Pointer to a variable of type *ClUint32T*, in which the current write offset is returned.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

Description:
This function is used to retrieve the current write offset of the message. This offset is updated on invoking the *clBufferNBytesWrite()* function or directly through *clBufferWriteOffsetSet()*.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferReadOffsetGet , clBufferDelete, clBufferReadOffsetSet , clBufferWriteOffsetSet, clBufferTrailerTrim , clBufferHeaderTrim

2.3.14 clBufferReadOffsetSet

clBufferReadOffsetSet

Synopsis:
Sets current read offset of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferReadOffsetSet(
                        CL_IN ClBufferHandleT messageHandle,
                        CL_IN ClInt32T newReadOffset,
                        CL_IN ClBufferSeekTypeT seekType);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function.

newReadOffset: (in) The offset at which the current read offset is to be set.

seekType: (in) This parameter accepts the following values:

- `CL_BUFFER_SEEK_SET`: The read offset is set to *newReadOffset* bytes from the beginning of the message.
- `CL_BUFFER_SEEK_CUR`: The read offset is set to its current location in addition to *newReadOffset* bytes.
- `CL_BUFFER_SEEK_END`: The read offset is set to the end of the message.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_INVALID_PARAMETER: On passing an invalid parameter.

Description:
This function is used to set the current read offset of the message. This offset is also automatically updated on invoking the *clBufferNBytesRead()* function.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferWriteOffsetGet , clBufferDelete, clBufferWriteOffsetSet, clBufferTrailerTrim , clBufferHeaderTrim, clBufferNBytesRead

2.3.15 clBufferWriteOffsetSet

clBufferWriteOffsetSet

Synopsis:
Sets current write offset of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferWriteOffsetSet(
                    CL_IN ClBufferHandleT messageHandle,
                    CL_IN ClInt32T newWriteOffset,
                    CL_IN ClBufferSeekTypeT seekType);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function whose current write offset is to be set.

newWriteOffset: (in) The offset at which the current write offset is to be set.

seekType: (in) This parameter can accept the following values:

- `CL_BUFFER_SEEK_SET`: The write offset is set to *newWriteOffset* bytes from the beginning of the message.
- `CL_BUFFER_SEEK_CUR`: The write offset is set to its current location in addition to *newWriteOffset* bytes.
- `CL_BUFFER_SEEK_END`: The write offset is set to the end of the message.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_INVALID_PARAMETER: If an invalid parameter is passed.

Description:
This function is used to set the current write offset of the message. This offset is also automatically updated on invoking the *clBufferNBytesWrite()*.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferWriteOffsetGet , clBufferDelete, clBufferReadOffsetSet, clBufferTrailerTrim , clBufferHeaderTrim, clBufferNBytesRead

2.3.16 clBufferHeaderTrim

clBufferHeaderTrim

Synopsis:
Trims the start of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferHeaderTrim (
CL_IN ClBufferHandleT messageHandle,
CL_IN ClUint32T numberOfBytes);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function.

numberOfBytes: (in) Number of bytes to be deleted from the start of the message.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_INVALID_PARAMETER: On passing an invalid parameter.

Description:
This function is used to delete a specific number of bytes from the start of the message. If a marker exists in the region being deleted, the marker becomes invalid and cannot be restored. If the read/write offset is in the region being deleted, the read/write offset will be set to zero (at the beginning of the message).

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferHeaderTrim, clBufferDelete

2.3.17 clBufferTrailerTrim

clBufferTrailerTrim

Synopsis:
Trims the tail of the message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferTrailerTrim(
                    CL_IN ClBufferHandleT messageHandle,
                    CL_IN ClUint32T numberOfBytes);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function.

numberOfBytes: (in) Number of bytes to delete from the trail of the message.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_INVALID_PARAMETER: On passing an invalid parameter.

Description:
This function is used to delete a specific number of bytes from tail of the message. If a marker exists in the region that is being deleted, the marker becomes invalid and cannot be restored. If the read/write offset is in the region being deleted, the read/write offset will be set to length of the message after delete, i.e, the end of the message.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferHeaderTrim, clBufferDelete

2.3.18 clBufferToBufferCopy

clBufferToMessageCopy

Synopsis:
Copies specific number of bytes from one message to another.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferToBufferCopy(
                    CL_IN ClBufferHandleT sourceMessage,
                    CL_IN ClUint32T sourceMessageOffset,
                    CL_IN ClBufferHandleT destinationMessage,
                    CL_IN ClUint32T numberOfBytes);
```

Parameters:

sourceMessage: (in) Handle to the message returned by the *clBufferCreate()* function, from which data is to be copied.

sourceMessageOffset: (in) Offset with respect to beginning of source message, from where the copying begins.

destinationMessage: (in) Handle to the message returned by the *clBufferCreate()* function, into which data is being copied.

numberOfBytes: (in) Number of bytes to be copied from the source message to the destination message.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_NOT_EXIST: If either of the messages is empty.

CL_ERR_INVALID_PARAMETER: On passing an invalid parameter.

CL_ERR_NO_MEMORY: On memory allocation failure.

Description:
This function is used to copy the specified number of bytes from the source message, starting at the source offset specified by the caller. The data will be copied into the destination message starting from current write-offset of destination message.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferReadOffsetGet, clBufferDelete, clBufferWriteOffsetGet, clBufferToBufferCopy, clBufferDuplicate

2.3.19 clBufferDuplicate

clBufferDuplicate

Synopsis:
Duplicates a message.

Header File:
clBufferApi.h

Syntax:

```
ClRcT clBufferDuplicate(
CL_IN ClBufferHandleT messageHandle,
CL_OUT ClBufferHandleT *pDuplicatedMessage);
```

Parameters:

messageHandle: (in) Handle to the message returned by *clBufferCreate()* function, which is to be duplicated.

pDuplicatedMessage: (out) Pointer to variable of type *ClBufferHandleT*, in which handle of the duplicate message is returned.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_NOT_EXIST: If either of the messages is empty.

CL_ERR_INVALID_PARAMETER: On passing an invalid parameter.

CL_ERR_NO_MEMORY: On memory allocation failure.

Description:
This function is used to make a copy of a message. All the markers are preserved in the new message being created.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferDelete, clBufferToBufferCopy, clBufferShare

2.3.20 clBufferShare

clBufferShare

Synopsis:
Shares a message.

Header File:
clBufferApi.h

Syntax:
```
ClRcT clBufferShare(
CL_IN ClBufferHandleT messageHandle,
CL_OUT ClBufferHandleT* pNewMessageHandle);
```

Parameters:
messageHandle: (in) Handle to the message returned by *clBufferCreate()* function, for which a reference is to be added.

pNewMessageHandle: (out) Pointer to variable of type ClBufferMessageHandleT, in which the reference to the message is returned.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

Description:
This function is used to indicate that a message is to be shared. When you invoke the *clBufferDelete()* function, the message will not be deleted until all the users referring to this message call the *clBufferDelete()* function.

Note:
After this function is invoked, the read and write offsets for the new reference handle will be the same as the original message and all markers will be preserved for the new reference.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferDelete, clBufferToBufferCopy, clBufferDuplicate

2.3.21 clBufferFlatten

clBufferFlatten

Synopsis:
Flattens message into a single buffer.

Header File:
clBufferApi.h

Syntax:
```
ClRcT clBufferFlatten(
CL_IN ClBufferHandleT messageHandle,
CL_OUT ClUint8T** ppFlattenBuffer);
```

Parameters:
messageHandle: (in) Handle to message returned by *clBufferCreate()* function.

*****ppFlattenBuffer:*** (out) The flattened buffer is returned at this location. This must be a valid pointer and cannot be NULL.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: On passing a NULL pointer.

CL_ERR_INVALID_HANDLE: On passing an invalid handle.

CL_ERR_NOT_INITIALIZED: If Buffer Management library is not initialized.

CL_ERR_NO_MEMORY: On memory allocation failure.

Description:
This function is used to flatten the message into a single buffer. The flattened buffer will be freed when the message is deleted. No reference to flat buffer will be available once the message is deleted.

Note:
This function must be invoked only once. After the successful invocation of this function, no other Buffer function must be invoked except *clBufferDelete()* function.

Library File:
ClBuffer

Related Function(s):
clBufferCreate, clBufferDelete

Index