Collection Manager Reference

Carbon > Data Management

2003-04-01

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Collection Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Collections.h

Overview

The Collection Manager implements an abstract data type that allows you to store multiple pieces of related information. This abstract data type is called a collection object. A collection object, or simply a collection, is an abstract data type that allows you to store information.

A collection is like an array in that it contains a number of individually accessible items. However, a collection offers some advantages over an array:

- A collection allows for a variable number of data items. You can add items to a collection or remove items from a collection during run time, and the Collection Manager automatically resizes the collection.
- A collection allows for variable-size items. Each item in a collection can contain data of any size.

A collection is also similar to a database, in that you can store information and retrieve it using a variety of search mechanisms.

The internal structure of a collection object is private–you must store information in a collection and retrieve information from it by providing a Collection Manager function with a reference to the collection. You use the functions provided by the Collection Manager to

- create and manipulate collection objects
- add information to a collection object
- retrieve information from a collection object
- store a collection object to disk and retrieve a collection object from disk

Carbon fully supports the Collection Manager.

Functions by Task

Adding and Replacing Items in a Collection

`AddCollectionItem` (page 11)
Adds a new item to a collection or to replace an existing item in a collection.

`ReplaceIndexedCollectionItem` (page 39)
Replaces the variable-length data of an item in a collection given the item's index.

Cloning and Copying Collection Objects

`CloneCollection` (page 14)
Clones a collection object—that is, increment its owner count.

`CopyCollection` (page 15)
Creates a copy of an existing collection.

`CountCollectionOwners` (page 16)
Determines the number of existing references to a collection object.

Counting Items in a Collection

`CountCollectionItems` (page 16)
Determines the total number of items in a collection.

`CountTaggedCollectionItems` (page 17)
Obtains the total number of items in a collection that have a specified collection tag.

Creating and Disposing of Collection Objects

`DisposeCollection` (page 18)
Disposes of a collection object.

`NewCollection` (page 35)
Creates a new, empty collection object.

Editing Item Attributes

`SetCollectionItemInfo` (page 43)
Edits the attributes of a specific collection item given the item's collection tag and collection ID.

`SetIndexedCollectionItemInfo` (page 44)
Edits the attributes of a specific collection item given the item's collection index.

Flattening and Unflattening a Collection

`FlattenCollection` (page 19)
Converts a collection object into a stream format suitable for storing and unflattening.

`FlattenPartialCollection` (page 21)
Converts a collection object into a stream format suitable for storage and unflattening.

`UnflattenCollection` (page 45)
Unflattens a collection that was flattened using the `FlattenCollection` or `FlattenPartialCollection` function.

Getting and Setting the Default Attributes for a Collection

`GetCollectionDefaultAttributes` (page 22)
Examines the default attributes of a collection object.

`SetCollectionDefaultAttributes` (page 42)
Alters the default attributes of a collection object.

Getting and Setting the Exception Procedure for a Collection

`GetCollectionExceptionProc` (page 23)
Obtains a pointer to the exception procedure installed in a specified collection.

`SetCollectionExceptionProc` (page 43)
Installs an exception procedure in a collection object.

Getting Information About a Collection Item

`GetCollectionItemInfo` (page 25)
Obtains information about a specific collection item given the item's collection tag and collection ID.

`GetIndexedCollectionItemInfo` (page 29)
Obtains information about a specific collection item given the item's collection index.

`GetTaggedCollectionItemInfo` (page 32)
Obtains information about a specific collection item given the item's collection tag and tag list position.

Getting Information About Collection Tags

`CollectionTagExists` (page 14)
Determines whether any of the items in a specified collection contain a specified collection tag.

`CountCollectionTags` (page 17)
Determines the number of distinct collection tags contained by the items of a specified collection.

`GetIndexedCollectionTag` (page 30)
Examines a specific collection tag contained in a collection.

Reading Collections From Resource Files

`GetNewCollection` (page 31)
Reads a collection in from a collection (`'cltn'`) resource.

Removing Items From a Collection

`EmptyCollection` (page 19)
Removes every item in a collection.

`PurgeCollection` (page 36)
Removes all items in a collection whose attributes match a specified pattern.

`PurgeCollectionTag` (page 37)
Removes all items with a specific collection tag from a collection.

`RemoveCollectionItem` (page 38)
Removes an item from a collection given the item's associated collection tag and collection ID.

`RemoveIndexedCollectionItem` (page 39)
Removes an item from a collection given the item's index.

Retrieving the Variable-Length Data From an Item

`GetCollectionItem` (page 24)
Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

`GetIndexedCollectionItem` (page 27)
Obtains a copy of the variable-length data associated with a collection item given the item's collection index.

`GetTaggedCollectionItem` (page 31)
Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and tag list position.

Working With Macintosh Memory Manager Handles

`AddCollectionItemHdl` (page 12)
Adds a new item to a collection or to replace an existing item in a collection, specifying the item's variable-length data using a handle rather than a pointer and a data size.

`FlattenCollectionToHdl` (page 20)
Flattens a collection into a Macintosh Memory Manager handle.

`GetCollectionItemHdl` (page 25)
Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

`GetIndexedCollectionItemHdl` (page 28)
Copies the variable-length data associated with a collection item into a Macintosh Memory Manager handle, given the item's collection index.

`ReplaceIndexedCollectionItemHdl` (page 41)
Replaces the variable-length data of an item in a collection given the item's collection index, specifying the item's new variable-length data using a handle rather than a pointer and a data size.

`UnflattenCollectionFromHdl` (page 46)
Unflattens a collection that was flattened using the `FlattenCollectionToHdl` utility function.

Working With Universal Procedure Pointers

`NewCollectionExceptionUPP` (page 35)
Creates a new universal procedure pointer (UPP) to an error-handling callback.

`InvokeCollectionExceptionUPP` (page 34)
Calls an error-handling callback.

`DisposeCollectionExceptionUPP` (page 18)
Disposes of a universal procedure pointer (UPP) to an error-handling callback.

`NewCollectionFlattenUPP` (page 36)
Creates a new universal procedure pointer (UPP) to a data-flattening callback.

`InvokeCollectionFlattenUPP` (page 34)
Calls a data-flattening callback.

`DisposeCollectionFlattenUPP` (page 19)
Disposes of a universal procedure pointer (UPP) to a data-flattening callback.

Retaining And Releasing

`RetainCollection` (page 42)
Increments the owner count (the number of existing references) for a collection object.

`GetCollectionRetainCount` (page 27)
Obtains the owner count (the number of existing references) for a collection object.

`ReleaseCollection` (page 38)
Decrements the owner count (the number of existing references) for a collection object.

Functions

AddCollectionItem

Adds a new item to a collection or to replace an existing item in a collection.

```
OSErr AddCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 itemSize,
   const void *itemData
);
```

Parameters

c
A reference to the collection you want to add the item to. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag
The collection tag you want to associate with the new item.

id
The collection ID you want to associate with the new item.

itemSize
The size in bytes of the item's variable-length data.

itemData
A pointer to the item's variable-length data.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
The `AddCollectionItem` function adds an item to the collection referenced by the `c` parameter. This new item contains

- the collection tag specified by the `tag` parameter
- the collection ID specified by the `id` parameter
- the attributes specified by the default attributes of the `c` collection
- the variable-length data specified by the `itemSize` and `itemData` parameters

This function copies the information pointed to by the `itemData` parameter into the new item; after calling this function, you may alter this information or free the memory pointed to by this parameter without affecting the collection.

If the `c` collection already contains an item with the same collection tag and collection ID as specified in the `tag` and `id` parameters, this function removes the original item and replaces it with the new one, unless the existing item is locked. If it is locked, this function returns a `collectionItemLockedErr` result code.

The `itemSize` parameter determines how many bytes of information this function copies into the new item. If you specify 0 for this parameter, or provide `NULL` for the `itemData` parameter, this function copies no information into the variable-length data of the new item, or removes the variable-length data if the item already exists.

To lock a collection item, use the functions `SetCollectionItemInfo` (page 43) and `SetIndexedCollectionItemInfo` (page 44).

To replace a collection item using the item's index (rather than the item's tag and ID), use the `ReplaceIndexedCollectionItem` (page 39) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

AddCollectionItemHdl

Adds a new item to a collection or to replace an existing item in a collection, specifying the item's variable-length data using a handle rather than a pointer and a data size.

```
OSErr AddCollectionItemHdl (
   Collection aCollection,
   CollectionTag tag,
   SInt32 id,
   Handle itemData
);
```

Parameters

aCollection

A reference to the collection you want to add the item to. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag you want to associate with the new item.

id

The collection ID you want to associate with the new item.

itemData

A Macintosh Memory Manager handle to the item's variable-length data. This function copies the information referenced by the `itemData` parameter into the new item; after calling this function, you may alter this information or free the memory referenced by this parameter without affecting the collection.

Return Value

A result code. See "Result Codes" (page 57). If the `aCollection` collection already contains an item with the same collection tag and collection ID as specified in the `tag` and `id` parameters, this function removes the variable-length data from the original item and replaces it with the new data, unless the existing item is locked. If it is locked, this function returns a `collectionItemLockedErr` result code.

Discussion

The `AddCollectionItemHdl` function adds an item to the collection referenced by the `aCollection` parameter. This new item contains:

- the collection tag specified by the `tag` parameter
- the collection ID specified by the `id` parameter
- the attributes specified by the default attributes of the aCollection collection
- the variable-length data specified by the `itemData` parameter

To add or replace a collection item using a pointer (rather than a handle) to the item's variable-length data, use the `AddCollectionItem` (page 11) function.

To replace a collection item using the item's collection index (rather than the item's collection tag and collection ID), use the `ReplaceIndexedCollectionItemHdl` (page 41) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CloneCollection

Clones a collection object—that is, increment its owner count.

```
Collection CloneCollection (
   Collection c
);
```

Parameters

c

A reference to the collection object you want to clone. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

A reference to the cloned collection. (This result is effectively a copy of the reference you provide in the `c` parameter. See the description of the `Collection` data type.

Discussion

Typically, you use this function to increment a collection object's owner count to represent a new reference to the collection object. For example, if you want two variables in your application to reference a single collection object, you can use this code to maintain the correct owner count:

```
firstReference = NewCollection();
secondReference = CloneCollection(firstReference);
```

Disposing of either reference (using the `DisposeCollection` function) simply decrements the collection's owner count. Disposing of the remaining reference decrements the owner count again and frees the memory associated with the collection.

To decrement the owner count of a collection object, use the `DisposeCollection` (page 18) function. To determine the owner count of an existing collection object, use the `CountCollectionOwners` (page 16) function.

To copy a collection object, use the `CopyCollection` (page 15) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CollectionTagExists

Determines whether any of the items in a specified collection contain a specified collection tag.

```
Boolean CollectionTagExists (
   Collection c,
   CollectionTag tag
);
```

Parameters

c

A reference to the collection object you want to search for a specific collection tag. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag to search for in the collection.

Return Value
`True` if the `c` collection contains any items that contain the specified tag.

Discussion
For information about data types related to collection tags, see `CollectionTag` (page 50).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

CopyCollection

Creates a copy of an existing collection.

```
Collection CopyCollection (
   Collection srcCollection,
   Collection dstCollection
);
```

Parameters

srcCollection

A reference to the collection object you want to copy. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

dstCollection

A reference to a collection object to contain the copied collection items. You may provide `NULL` for this parameter to request that the Collection Manager create a new collection object to hold the copied information.

Return Value
A reference to the collection object containing the copied information. See the description of the `Collection` data type.

Discussion
The `CopyCollection` function copies all of the information (except the owner count and exception procedure) from the collection object referenced by the `srcCollection` parameter into the collection object referenced by the `dstCollection` parameter.

If you specify `NULL` for the `dstCollection` parameter, this function creates a new collection object to copy the information into. (This function does not return an error code; it returns `NULL` if it cannot create a new collection object.)

To clone a collection object, use the `DisposeCollection` (page 18) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

CountCollectionItems

Determines the total number of items in a collection.

```
SInt32 CountCollectionItems (
   Collection c
);
```

Parameters

c

A reference to the collection object whose items you want to count. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

The total number of items in the `c` collection.

Discussion

To count the items in a collection that have a specified collection tag, use the `CountTaggedCollectionItems` (page 17) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CountCollectionOwners

Determines the number of existing references to a collection object.

```
SInt32 CountCollectionOwners (
   Collection c
);
```

Parameters

c

The collection object whose owner count you want to determine. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

The owner count of the collection object.

Discussion

To increment the owner count of a collection object, use the `CloneCollection` (page 14) function. To decrement the owner count of a collection object, use the `DisposeCollection` (page 18) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CountCollectionTags

Determines the number of distinct collection tags contained by the items of a specified collection.

```
SInt32 CountCollectionTags (
   Collection c
);
```

Parameters

c

A reference to the collection object whose collection tags you want to count. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

The number of distinct collection tags contained by the items of the `c` collection.

Discussion

For information about data types related to collection tags, see `CollectionTag` (page 50).

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CountTaggedCollectionItems

Obtains the total number of items in a collection that have a specified collection tag.

```
SInt32 CountTaggedCollectionItems (
   Collection c,
   CollectionTag tag
);
```

Parameters

c

A reference to the collection object whose items you want to count. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the items you want to count.

Return Value

The total number of items in the `c` collection whose collection tags match the value specified in the `tag` parameter.

Discussion

To count all of the items in a collection, use the `CountCollectionItems` (page 16) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

DisposeCollection

Disposes of a collection object.

```
void DisposeCollection (
   Collection c
);
```

Parameters

c

A reference to the collection object you want to dispose of. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Discussion

The `DisposeCollection` function decrements the owner count of the collection object referenced by the `c` parameter. If the resulting owner count is 0, this function releases the memory occupied by the collection object, and the collection object reference contained in the `c` parameter becomes invalid.

To create a new collection object, use the `NewCollection` (page 35) function.

To increment the owner count of a collection object, use the `CloneCollection` (page 14) function. To determine the owner count of an existing collection object, use the `CountCollectionOwners` (page 16) function

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

DisposeCollectionExceptionUPP

Disposes of a universal procedure pointer (UPP) to an error-handling callback.

```
void DisposeCollectionExceptionUPP (
   CollectionExceptionUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `CollectionExceptionProcPtr` (page 47) for more information.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

DisposeCollectionFlattenUPP

Disposes of a universal procedure pointer (UPP) to a data-flattening callback.

```
void DisposeCollectionFlattenUPP (
   CollectionFlattenUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `CollectionFlattenProcPtr` (page 48) for more information.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

EmptyCollection

Removes every item in a collection.

```
void EmptyCollection (
   Collection c
);
```

Parameters

c

A reference to the collection object you want to empty. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Discussion

This function removes every item in the collection referenced by the `c` parameter. This function provides the fastest mechanism for emptying a collection.

To remove all of the items in a collection whose attributes match a specified pattern, use the `PurgeCollection` (page 36) function.

To remove all of the items in a collection with a specified collection tag, use the `PurgeCollectionTag` (page 37) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

FlattenCollection

Converts a collection object into a stream format suitable for storing and unflattening.

```
OSErr FlattenCollection (
   Collection c,
   CollectionFlattenUPP flattenProc,
   void *refCon
);
```

Parameters

c

A reference to the collection that you want to flatten. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattenProc

A pointer to a callback function you provide to process the flattened stream of bytes.

refCon

A pointer to the reference constant that you want the Collection Manager to pass to your callback function each time that it calls the callback function. You can use this parameter as a pointer to a structure containing information your callback function needs to process the blocks of flattened data.

Return Value
A result code. See "Result Codes" (page 57). This function can return any error returned by the callback function.

Discussion
You could, for example, use this function to copy a collection onto the Clipboard so that it could be pasted into another application.

The `FlattenCollection` function flattens into a stream of bytes the collection you specify with the `c` parameter. As this function flattens the collection, it repeatedly calls the callback function you specify using the `flattenProc` parameter. Each time it calls this function, it provides the callback function with a pointer to a block of memory containing flattened data. It continues to call this function until it has flattened the entire collection. Your callback function can process the flattened data in a number of ways: it could copy the flattened data into a handle-based block of memory, it could write the flattened data to disk, and so on.

When flattening the `c` collection, this function includes only the collection items whose persistence attribute is set.

To create a flattened collection that includes only those collection items whose attributes match a specified pattern, use the `FlattenPartialCollection` (page 21) function.

To unflatten a flattened collection, use the `UnflattenCollection` (page 45) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

FlattenCollectionToHdl

Flattens a collection into a Macintosh Memory Manager handle.

```
OSErr FlattenCollectionToHdl (
   Collection aCollection,
   Handle flattened
);
```

Parameters

aCollection

The collection that you want to flatten into a handle. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattened

A handle to contain the flattened data. You must provide a valid Macintosh Memory Manager handle in this parameter. You may specify a handle of size 0; this function resizes the handle as necessary to hold the flattened data.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

This function flattens the collection referenced by the `aCollection` parameter into a block of memory referenced by the handle you provide in the `flattened` parameter.

To flatten a collection directly to disk, use the `FlattenCollection` (page 19) function.

To unflatten a collection from a block of memory referenced by a handle, use the `UnflattenCollectionFromHdl` (page 46) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

FlattenPartialCollection

Converts a collection object into a stream format suitable for storage and unflattening.

```
OSErr FlattenPartialCollection (
   Collection c,
   CollectionFlattenUPP flattenProc,
   void *refCon,
   SInt32 whichAttributes,
   SInt32 matchingAttributes
);
```

Parameters

c

The collection that you want to flatten. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattenProc

A pointer to a function to write data.

refCon

A reference constant that you want the Collection Manager to pass to your flatten function each time it calls the flatten function. You can use this parameter as a pointer to a structure containing information your callback function needs to process the blocks of flattened data.

whichAttributes

A mask indicating which attributes you want to test.

matchingAttributes

An `SInt32` word containing the attribute values you want to match.

Return Value
A result code. See "Result Codes" (page 57). This function can return any error returned by the callback function.

Discussion
With this function, you can include in the flattened collection only those items whose attributes match a specified pattern.

The `FlattenPartialCollection` function flattens into a stream of bytes the collection you specify with the `c` parameter. It includes only the collection items whose attributes specified by the `whichAttributes` parameter match the values specified by the `matchingAttributes` parameter.

As this function flattens the collection, it repeatedly calls the callback function you specify using the `flattenProc` parameter. Each time it calls this function, it provides the callback function with a pointer to a block of memory containing flattened data. It continues to call this function until it has flattened the entire collection. Your callback function can process the flattened data in a number of ways: it could copy the flattened data into a handle-based block of memory, it could write the flattened data to disk, and so on.

When flattening the c collection, this function includes only the collection items whose persistence attribute is set, regardless of the values you provide in the `whichAttributes` and `matchingAttributes` parameters.

To create a flattened collection that includes every item in a collection, use the `FlattenCollection` (page 19) function.

To unflatten a flattened collection, use the `UnflattenCollection` (page 45) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionDefaultAttributes

Examines the default attributes of a collection object.

```
SInt32 GetCollectionDefaultAttributes (
   Collection c
);
```

Parameters

c

A reference to the collection object whose default attributes you want to determine. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value
An `SInt32` word containing the bit flags that make up the collection's default attributes.

Discussion
To change the attributes of a collection object, use the `SetCollectionDefaultAttributes` (page 42) function.

To examine the attributes of a specific item in a collection, use `GetCollectionItemInfo` (page 25), `GetIndexedCollectionItemInfo` (page 29), and `GetTaggedCollectionItemInfo` (page 32)

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionExceptionProc

Obtains a pointer to the exception procedure installed in a specified collection.

```
CollectionExceptionUPP GetCollectionExceptionProc (
   Collection c
);
```

Parameters

c

A reference to the collection object whose exception procedure you want to determine. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value
A pointer to the exception procedure installed in the `c` collection object. See the description of the `CollectionExceptionUPP` data type.

Discussion
To install a new exception procedure in a collection object, use the `SetCollectionExceptionProc` (page 43) function.

For more information about exception procedures, see `CollectionExceptionProcPtr` (page 47).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionItem

Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

```
OSErr GetCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 *itemSize,
   void *itemData
);
```

Parameters

c

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item whose data you want to retrieve.

id

The collection ID associated with the item whose data you want to retrieve.

itemSize

A pointer to an `SInt32` value indicating the number of bytes of data you want returned in the `itemData` parameter. On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you want to copy all the specified item's variable-length data and you do not want to determine the size of this data. You may specify a value for the `itemSize` parameter that is greater than the actual number of bytes in the specified item's variable-length data however, this function never returns in the `itemData` parameter more data than contained in the specified item's variable-length data.

itemData

A pointer to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

If you do not know the size of the item you want to retrieve, you typically call this function twice. The first time you provide a pointer in the `itemSize` parameter to determine the size of the specified item's data and you specify `dontWantData` for the `itemData` parameter. Then you allocate a memory block large enough to hold a copy of the item's data. Then you call the function a second time. This time you specify the constant `dontWantSize` for the `itemSize` parameter and provide a pointer to the allocated memory block for the `itemData` parameter. The function then copies the data into the allocated block of memory.

To retrieve the data associated with a collection item given its collection index (rather than its collection tag and ID), use the `GetIndexedCollectionItem` (page 27) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionItemHdl

Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

```
OSErr GetCollectionItemHdl (
   Collection aCollection,
   CollectionTag tag,
   SInt32 id,
   Handle itemData
);
```

Parameters

aCollection
A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag
The collection tag associated with the item whose data you want to retrieve.

id
The collection ID associated with the item whose data you want to retrieve.

itemData
A handle to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You must provide a valid Macintosh Memory Manager handle for this function to copy the data into.You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
You specify a collection object using the `aCollection` parameter and you specify an item in that collection using the `tag` and `id` parameters.

To retrieve the data associated with a collection item into a block of memory referenced by a pointer (rather than a handle), use the `GetCollectionItem` (page 24) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionItemInfo

Obtains information about a specific collection item given the item's collection tag and collection ID.

```
OSErr GetCollectionItemInfo (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 *index,
   SInt32 *itemSize,
   SInt32 *attributes
);
```

Parameters

c

A reference to the collection object containing the item you want to obtain information about. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item you want to obtain information about.

id

The collection ID associated with the item you want to obtain information about.

index

On return, this value represents the collection index of the specified item. You may specify the constant `dontWantIndex` for this parameter if you do not want to determine the specified item's collection index.

itemSize

On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you do not want to determine the size of this data.

attributes

On return, this value contains a copy of the attributes associated with the specified item. You may specify the constant `dontWantAttributes` for this parameter if you do not want a copy of the item's attributes.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

This function returns information in the `index`, `itemSize`, and `attributes` parameters:

- If you provide a pointer in the `index` parameter, the function uses this parameter to return the collection index of the specified item. Once you have determined an item's collection index, you can use it to specify the item when calling Collection Manager functions, rather than using the item's collection tag and collection ID. Specifying collection items using their collection index, rather than using the item's collection tag and collection ID, generally results in improved performance.
- If you provide a pointer in the `itemSize` parameter, the function uses this parameter to return the size in bytes of the variable-length data associated with the specified collection item.
- If you provide a pointer in the `attributes` parameter, the function uses this parameter to return a copy of the attributes associated with the specified collection item.

To obtain information about a collection item using the collection index to specify the item, use the `GetIndexedCollectionItemInfo` (page 29) function.

To obtain information about a collection item using the tag and `whichItem` parameters to specify the item, use the `GetTaggedCollectionItemInfo` (page 32) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionRetainCount

Obtains the owner count (the number of existing references) for a collection object.

```
ItemCount GetCollectionRetainCount (
   Collection c
);
```

Parameters

c

Discussion
This function performs the same operation as `CountCollectionOwners` (page 16), but follows the preferred naming conventions for Carbon and Core Foundation functions.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.1 and later.

Declared In
`Collections.h`

GetIndexedCollectionItem

Obtains a copy of the variable-length data associated with a collection item given the item's collection index.

```
OSErr GetIndexedCollectionItem (
   Collection c,
   SInt32 index,
   SInt32 *itemSize,
   void *itemData
);
```

Parameters

c

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item whose data you want to retrieve.

itemSize

A pointer to an SInt32 value indicating the number of bytes of data you want returned in the itemData parameter. On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant dontWantSize for this parameter to indicate that you want to copy all of the specified item's variable-length data and you do not want to determine the size of this data. You may specify a value for the itemSize parameter that is greater than the actual number of bytes in the specified item's variable-length data however, this function never returns in the itemData parameter more data than contained in the specified item's variable-length data.

itemData

A pointer to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You may specify the constant dontWantData for this parameter if you do not want a copy of the item's data.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

If you do not know the size of the item you want to retrieve, you typically call this function twice. The first time you provide a pointer in the itemSize parameter to determine the size of the specified item's data and you specify the constant dontWantData for the itemData parameter. Then you allocate a memory block large enough to hold a copy of the item's data. Then you call the function a second time. This time you specify the constant dontWantSize for the itemSize parameter and provide a pointer to the allocated memory block for the itemData parameter. The function then copies the data into the allocated block of memory.

To retrieve the data associated with a collection item given its collection tag and ID (rather than its collection index), use the GetCollectionItem (page 24) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

Collections.h

GetIndexedCollectionItemHdl

Copies the variable-length data associated with a collection item into a Macintosh Memory Manager handle, given the item's collection index.

```
OSErr GetIndexedCollectionItemHdl (
   Collection aCollection,
   SInt32 index,
   Handle itemData
);
```

Parameters

aCollection

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item whose data you want to retrieve.

itemData

A handle to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
To retrieve the data associated with a collection item into a block of memory referenced by a pointer (rather than a handle), use the `GetCollectionItem` (page 24) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetIndexedCollectionItemInfo

Obtains information about a specific collection item given the item's collection index.

```
OSErr GetIndexedCollectionItemInfo (
   Collection c,
   SInt32 index,
   CollectionTag *tag,
   SInt32 *id,
   SInt32 *itemSize,
   SInt32 *attributes
);
```

Parameters

c

A reference to the collection object containing the item you want to obtain information about. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item you want to obtain information about.

tag

A pointer to a collection tag. On return, the collection tag associated with the specified item. You may specify the constant `dontWantTag` for this parameter if you do not want to determine the specified item's collection tag.

id

A pointer to an `SInt32` value. On return, the collection ID associated with the specified item. You may specify the constant `dontWantId` for this parameter if you do not want to determine the specified item's collection ID.

itemSize

A pointer to an `SInt32` value. On return, this value indicates the size in bytes of the data associated with the specified item. You may specify the constant `dontWantSize` for this parameter if you do not want to determine the specified item's data size.

attributes

A pointer to an `SInt32` value. On return, this value contains a copy of the attributes associated with the specified item. You may specify the constant `dontWantAttributes` for this parameter if you do not want a copy of the item's attributes.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
To obtain information about a collection item using the collection tag and collection ID to specify the item, use the `GetCollectionItemInfo` (page 25) function.

To obtain information about a collection item using the collection tag and tag list position to specify the item, use the `GetTaggedCollectionItemInfo` (page 32) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetIndexedCollectionTag

Examines a specific collection tag contained in a collection.

```
OSErr GetIndexedCollectionTag (
   Collection c,
   SInt32 tagIndex,
   CollectionTag *tag
);
```

Parameters

c

The collection from which to obtain a specific collection tag. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tagIndex

The position of the desired collection tag in the `c` collection's list of distinct collection tags.

tag

A pointer to a collection tag. On return, the collection tag that lies at the specified position in the list of distinct collection tags contained in the `c` collection.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
Each collection object contains a number of distinct collection tags. By sequentially incrementing the value of the `tagIndex` parameter from 1 to the result of the `CountCollectionTags` (page 17) function, you can use this function to determine every collection tag contained in a collection.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetNewCollection

Reads a collection in from a collection (`'cltn'`) resource.

```
Collection GetNewCollection (
   SInt16 collectionID
);
```

Parameters

collectionID

The resource ID associated with the collection resource from which you want to create the new collection object.

Return Value
A reference to the new collection object. If this function does not find a collection resource with the specified resource ID, it returns `NULL` as the function result. See the description of the `Collection` data type.

Discussion
This function searches the current resource file path for a collection (`'cltn'`) resource with the resource ID specified by the `collectionID` parameter. If it finds such a resource, this function creates a new collection object, initializes it with the information stored in the resource, and returns a reference to it as the function result.

You can use the `MemError` and `ResError` functions to check for other errors after calling this function.

For information about collection resources, see 'cltn'.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetTaggedCollectionItem

Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and tag list position.

```
OSErr GetTaggedCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 whichItem,
   SInt32 *itemSize,
   void *itemData
);
```

Parameters

c

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item whose data you want to retrieve.

whichItem

The tag list position associated with the specific item.

itemSize

A pointer to an `SInt32` value indicating the number of bytes of data you want returned in the `itemData` parameter. On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you want to copy all of the specified item's variable-length data and you do not want to determine the size of this data.

itemData

A pointer to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
Remember that a tag list position is the sequential index that determines an item given a specific collection tag. For example:

- A `whichItem` value of 1 indicates the first item with the specified tag.
- A `whichItem` value of 2 indicates the second item with the specified tag.

By sequentially incrementing the whichItem parameter, you can use this function to step through all of the items in a collection without knowing their collection IDs.

If you do not know the size of the item you want to retrieve, you typically call this function twice. The first time you provide a pointer in the `itemSize` parameter to determine the size of the specified item's data and you specify the constant `dontWantData` for the `itemData` parameter. Then you allocate a memory block large enough to hold a copy of the item's data. Then you call the function a second time. This time you specify the constant `dontWantSize` for the `itemSize` parameter and provide a pointer to the allocated memory block for the `itemData` parameter. The function then copies the data into the allocated block of memory.

To retrieve the data associated with a collection item given its collection tag and ID, use the `GetCollectionItem` (page 24) function.

To retrieve the data associated with a collection item given its collection index, use the `GetIndexedCollectionItem` (page 27) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetTaggedCollectionItemInfo

Obtains information about a specific collection item given the item's collection tag and tag list position.

```
OSErr GetTaggedCollectionItemInfo (
   Collection c,
   CollectionTag tag,
   SInt32 whichItem,
   SInt32 *id,
   SInt32 *index,
   SInt32 *itemSize,
   SInt32 *attributes
);
```

Parameters

c

A reference to the collection object containing the item you want to obtain information about. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item you want to obtain information about.

whichItem

The tag list position of the item you want to obtain information about.

id

A pointer to an `SInt32` value. On return, this value represents the collection ID associated with the specified item. You may specify the constant `dontWantId` for this parameter if you do not want to determine the specified item's collection ID.

index

A pointer to an `SInt32` value. On return, this value represents the collection index of the specified item. You may specify the constant `dontWantIndex` for this parameter if you do not want to determine the specified item's collection index.

itemSize

A pointer to an `SInt32` value. On return, this value indicates the size in bytes of the data associated with the specified item. You may specify the constant `dontWantSize` for this parameter if you do not want to determine the specified item's data size.

attributes

A pointer. On return, this value contains a copy of the attributes associated with the specified item. You may specify the constant `dontWantAttributes` for this parameter if you do not want a copy of the item's attributes.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

Remember that a collection tag and a tag list position uniquely identify a collection item. The tag list position indicates where the collection item would lie in a list made up of all the collection items with the same collection tag. For example:

- A `whichItem` value of 1 indicates the first item with the specified tag.
- A `whichItem` value of 2 indicates the second item with the specified tag.

By sequentially incrementing the `whichItem` parameter, you can use this function to step through all of the items in a collection that share a collection tag without knowing their collection IDs.

To obtain information about a collection item using the collection tag and collection ID to specify the item, use the `GetCollectionItemInfo` (page 25) function.

To obtain information about a collection item using the collection index to specify the item, use the `GetIndexedCollectionItemInfo` (page 29) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

InvokeCollectionExceptionUPP

Calls an error-handling callback.

```
OSErr InvokeCollectionExceptionUPP (
   Collection c,
   OSErr status,
   CollectionExceptionUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeCollectionExceptionUPP`, as the system calls your error-handling callback function for you. See the callback `CollectionExceptionProcPtr` (page 47) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

InvokeCollectionFlattenUPP

Calls a data-flattening callback.

```
OSErr InvokeCollectionFlattenUPP (
   SInt32 size,
   void *data,
   void *refCon,
   CollectionFlattenUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeCollectionFlattenUPP`, as the system calls your data-flattening callback function for you. See the callback `CollectionFlattenProcPtr` (page 48) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

NewCollection

Creates a new, empty collection object.

```
Collection NewCollection (
   void
);
```

Return Value
A reference to the newly created collection object. The new collection contains no items and has an owner count of 1. The `NewCollection` function does not return an error code; it returns `NULL` if it cannot create a new collection object. See the description of the `Collection` data type.

Discussion
The `NewCollection` function allocates memory for a new collection object, initializes it, and returns a reference to it.

To create a copy of an existing collection object, use the `CopyCollection` (page 15) function.

Special Considerations

You are responsible for disposing of collection objects that you create with this function when you no longer need them. To dispose of a collection object, use the `DisposeCollection` (page 18) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

NewCollectionExceptionUPP

Creates a new universal procedure pointer (UPP) to an error-handling callback.

```
CollectionExceptionUPP NewCollectionExceptionUPP (
   CollectionExceptionProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your error-handling callback.

Return Value
On return, a UPP to the error-handling callback. See the description of the `CollectionExceptionUPP` data type.

Discussion
See the callback `CollectionExceptionProcPtr` (page 47) for more information.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

NewCollectionFlattenUPP

Creates a new universal procedure pointer (UPP) to a data-flattening callback.

```
CollectionFlattenUPP NewCollectionFlattenUPP (
   CollectionFlattenProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your data-flattening callback.

Return Value
On return, a UPP to the data-flattening callback. See the description of the `CollectionFlattenUPP` data type.

Discussion
See the callback `CollectionFlattenProcPtr` (page 48) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

PurgeCollection

Removes all items in a collection whose attributes match a specified pattern.

```
void PurgeCollection (
   Collection c,
   SInt32 whichAttributes,
   SInt32 matchingAttributes
);
```

Parameters

c
A reference to the collection object containing the items you want to remove. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

whichAttributes
A mask indicating which attributes you want to test. You should set the bits of the parameter that correspond to the attributes you want to test.

matchingAttributes
An `SInt32` word containing the values of the attributes you want to match.

Discussion
The `PurgeCollection` function removes from the c collection any items whose attributes match the criteria you specify in the `whichAttributes` and `matchingAttributes` parameters.

This function compares the specified attributes of each item in the c collection with the corresponding attributes in the `matchingAttributes` parameter. If the values of all the specified attributes match, the function removes the item. To avoid purging locked items, you should clear the lock attribute in the `whichAttributes` and `matchingAttributes` parameters.

To remove all of the items in a collection with a specified collection tag, use the `PurgeCollectionTag` (page 37) function.

To remove every item in a collection, use the `EmptyCollection` (page 19) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

PurgeCollectionTag

Removes all items with a specific collection tag from a collection.

```
void PurgeCollectionTag (
   Collection c,
   CollectionTag tag
);
```

Parameters

c

A reference to the collection object containing the items you want to remove. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the items to remove.

Discussion
The `PurgeCollectionTag` function removes from the `c` collection all items whose collection tag matches the value of the `tag` parameter. This function removes locked and unlocked items.

To remove all of the items in a collection whose attributes match a specified pattern, use the `PurgeCollection` (page 36) function.

To remove every item in a collection, use the `EmptyCollection` (page 19) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

ReleaseCollection

Decrements the owner count (the number of existing references) for a collection object.

```
OSStatus ReleaseCollection (
   Collection c
);
```

Parameters

c

Return Value

A result code. See "Result Codes" (page 57).

Discussion

This function performs the same operation as `DisposeCollection` (page 18), but follows the preferred naming conventions for Carbon and Core Foundation functions.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.1 and later.

Declared In

`Collections.h`

RemoveCollectionItem

Removes an item from a collection given the item's associated collection tag and collection ID.

```
OSErr RemoveCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 id
);
```

Parameters

c

A reference to the collection object from which you want to remove the item. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item you want to remove.

id

The collection ID associated with the item you want to remove.

Return Value

A result code. See "Result Codes" (page 57). If the `c` collection does not contain an item whose collection tag and collection ID match the values in the `tag` and `id` parameters, this function returns a `collectionItemNotFoundErr` result code.

Discussion

The `RemoveCollectionItem` function removes the item specified by the `tag` and `id` parameters from the collection referenced by the `c` parameter. This function removes the specified item even if its lock attribute is set.

To remove a collection item using the item's index (rather than the item's tag and ID), use the `RemoveIndexedCollectionItem` (page 39) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

RemoveIndexedCollectionItem

Removes an item from a collection given the item's index.

```
OSErr RemoveIndexedCollectionItem (
   Collection c,
   SInt32 index
);
```

Parameters

c

A reference to the collection object from which you want to remove the item. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index of the item you want to remove.

Return Value
A result code. See "Result Codes" (page 57). If the `c` collection does not contain an item whose collection index matches the values in the `index` parameter, this function returns a collectionIndexRangeErr result code.

Discussion
The `RemoveIndexedCollectionItem` function removes the item specified by the `index` parameter from the collection referenced by the `c` parameter. This function removes the specified item even if its lock attribute is set.

To remove a collection item using the item's tag and ID (rather than the item's index), use the `RemoveCollectionItem` (page 38) function.

To replace an item in a collection, use the function `ReplaceIndexedCollectionItem` (page 39).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

ReplaceIndexedCollectionItem

Replaces the variable-length data of an item in a collection given the item's index.

```
OSErr ReplaceIndexedCollectionItem (
   Collection c,
   SInt32 index,
   SInt32 itemSize,
   const void *itemData
);
```

Parameters

c

A reference to the collection containing the item you want to replace. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item to replace.

itemSize

The item's size. The `itemSize` parameter determines how many bytes of information this function copies into the new item. If you specify 0 for this parameter, or provide `NULL` for the `itemData` parameter, this function copies no information into the variable-length data of the new item, or removes the variable-length data if the item already exists.

itemData

A pointer to the item's data. This function copies the information pointed to by the `itemData` parameter into the new item; after calling this function, you may alter this information or free the memory pointed to by this parameter without affecting the collection.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

You specify which item to replace using the `index` parameter. If the c collection does not contain an item whose collection index matches the value of the `index` parameter, this function returns a `collectionIndexRangeErr` result code.

If the c collection does contain an item with the specified index, this function replaces that item with a new item (if the existing item is not locked—if it is, this function returns a `collectionItemLockedErr` result code). The new item contains

- the same collection tag as the original item
- the same collection ID as the original item
- the same attributes as the original item
- the variable-length data specified by the `itemSize` and `itemData` parameters

To lock a collection item, use the functions `SetCollectionItemInfo` (page 43) and `SetIndexedCollectionItemInfo` (page 44).

To replace a collection item using the item's tag and ID (rather than the item's index), use the `AddCollectionItem` (page 11) function.

To remove an item from a collection, use the functions `RemoveCollectionItem` (page 38), `RemoveIndexedCollectionItem` (page 39), `PurgeCollection` (page 36), `PurgeCollectionTag` (page 37), and `EmptyCollection` (page 19).

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

ReplaceIndexedCollectionItemHdl

Replaces the variable-length data of an item in a collection given the item's collection index, specifying the item's new variable-length data using a handle rather than a pointer and a data size.

```
OSErr ReplaceIndexedCollectionItemHdl (
   Collection aCollection,
   SInt32 index,
   Handle itemData
);
```

Parameters

aCollection

A reference to the collection containing the item you want to replace. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item you want to replace.

itemData

A Macintosh Memory Manager handle to the new variable-length data. This function copies the information referenced by the `itemData` parameter into the collection item; after calling this function, you may alter this information or free the memory referenced by this parameter without affecting the collection.

Return Value
A result code. See "Result Codes" (page 57). If the `aCollection` collection does not contain an item whose collection index matches the value of the `index` parameter, this function returns a `collectionIndexRangeErr` result code.

Discussion
If the `aCollection` collection does contain an item with the specified index, this function replaces the data in that item with new data (if the existing item is not locked—if it is, this function returns a `collectionItemLockedErr` result code). The resulting item contains

- the same collection tag as the original item
- the same collection ID as the original item
- the same attributes as the original item
- the variable-length data specified by the `itemData` parameter

To replace a collection item using a pointer (rather than a handle) to the item's variable-length data, use the `ReplaceIndexedCollectionItem` (page 39) function.

To replace a collection item using the item's collection tag and collection ID (rather than the item's collection index), use the `AddCollectionItemHdl` (page 12) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

RetainCollection

Increments the owner count (the number of existing references) for a collection object.

```
OSStatus RetainCollection (
   Collection c
);
```

Parameters

c

Return Value
A result code. See "Result Codes" (page 57).

Discussion
This function performs the same operation as `CloneCollection` (page 14), but follows the preferred naming conventions for Carbon and Core Foundation functions.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.1 and later.

Declared In
`Collections.h`

SetCollectionDefaultAttributes

Alters the default attributes of a collection object.

```
void SetCollectionDefaultAttributes (
   Collection c,
   SInt32 whichAttributes,
   SInt32 newAttributes
);
```

Parameters

c

A reference to the collection object whose default attributes you want to alter. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

whichAttributes

A mask indicating which bit flags in the c collection's default attributes you want to alter. For every bit in the `whichAttributes` parameter, this function takes one of two actions:

- If the bit is set, this function copies the value of the corresponding bit from the `newAttributes` parameter into the corresponding bit of the default attributes of the c collection.
- If the bit is not set, the corresponding bit of the c collection's default attributes remains unchanged.

newAttributes

The new values for the bit flags.

Discussion

To examine the attributes of a collection object, use the `GetCollectionDefaultAttributes` (page 22) function.

To change the attributes of a specific item in a collection, use the functions `SetCollectionItemInfo` (page 43) and `SetIndexedCollectionItemInfo` (page 44).

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

SetCollectionExceptionProc

Installs an exception procedure in a collection object.

```
void SetCollectionExceptionProc (
   Collection c,
   CollectionExceptionUPP exceptionProc
);
```

Parameters

c

A reference to the collection object whose exception procedure you want to change. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

exceptionProc

A pointer to the new exception procedure.

Discussion

The `SetCollectionExceptionProc` function copies the function pointer from the `exceptionProc` parameter into the collection object referenced by the `c` parameter.

To obtain a pointer to an existing exception procedure in a collection object, use the `GetCollectionExceptionProc` (page 23) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

SetCollectionItemInfo

Edits the attributes of a specific collection item given the item's collection tag and collection ID.

```
OSErr SetCollectionItemInfo (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 whichAttributes,
   SInt32 newAttributes
);
```

Parameters

c

A reference to the collection object containing the item whose attributes you want to edit. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item whose attributes you want to edit.

id

The collection ID associated with the item whose attributes you want to edit.

whichAttributes

A mask indicating which attributes you want to edit.

newAttributes

An `SInt32` word containing the new settings for the attributes.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
This function copies bit values from the `newAttributes` parameter to the attributes associated with the specified item.

This function uses the `whichAttributes` parameter to determine which bits to copy. For every bit in the `whichAttributes` parameter, this function takes one of two actions:

- If the bit is set, this function copies the value of the corresponding bit from the `newAttributes` parameter into the corresponding bit of the attributes associated with the specified item.
- If the bit is not set, the corresponding bit of the specified item's attributes remains unchanged.

The `whichAttributes` parameter allows you to change the values of specific bits in the specified item's attributes without affecting the values of other bits.

To obtain information about a collection item using the collection index to specify the item, use the `SetIndexedCollectionItemInfo` (page 44) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

SetIndexedCollectionItemInfo

Edits the attributes of a specific collection item given the item's collection index.

```
OSErr SetIndexedCollectionItemInfo (
   Collection c,
   SInt32 index,
   SInt32 whichAttributes,
   SInt32 newAttributes
);
```

Parameters

c

A reference to the collection object containing the item whose attributes you want to edit. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index of the item whose attributes you want to edit.

whichAttributes

A mask indicating which attributes you want to edit.

newAttributes

An `SInt32` word containing the new settings for the attributes.

Return Value

A result code. See "Result Codes" (page 57).

Discussion

The `SetIndexedCollectionItemInfo` function copies bit values from the `newAttributes` parameter to the attributes associated with the specified item.

This function uses the `whichAttributes` parameter to determine which bits to copy. For every bit in the `whichAttributes` parameter, this function takes one of two actions:

- If the bit is set, this function copies the value of the corresponding bit from the `newAttributes` parameter into the corresponding bit of the attributes associated with the specified item.
- If the bit is not set, the corresponding bit of the specified item's attributes remains unchanged.

The `whichAttributes` parameter allows you to change the values of specific bits in the specified item's attributes without affecting the values of other bits.

To edit the attributes of collection item using the collection tag and collection ID (rather than the collection index) to specify the item, use the `SetCollectionItemInfo` (page 43) function.

To examine the attributes of a collection item, use the functions `GetCollectionItemInfo` (page 25), `GetIndexedCollectionItemInfo` (page 29), and `GetTaggedCollectionItemInfo` (page 32).

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

UnflattenCollection

Unflattens a collection that was flattened using the `FlattenCollection` or `FlattenPartialCollection` function.

```
OSErr UnflattenCollection (
   Collection c,
   CollectionFlattenUPP flattenProc,
   void *refCon
);
```

Parameters

c

A reference to the collection object you want to create from the flattened data. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattenProc

A pointer to a function to read in flattened data.

refCon

A reference constant that you want the Collection Manager to pass to your callback function each time it calls the callback function. You can use this parameter as a pointer to a structure containing information your callback function needs when reading the blocks of flattened data.

Return Value

A result code. See "Result Codes" (page 57). This function can return any error returned by the callback function.

Discussion

The `UnflattenCollection` function unflattens a stream of bytes into the collection object you specify with the `c` parameter.

As this function unflattens the collection, it repeatedly calls the callback function you specify using the `flattenProc` parameter. Each time it calls this function, it provides the callback function with a pointer to a block of memory and a requested size. The callback function is responsible for reading the next set of bytes from the flattened byte stream and copying the data into the block of memory.

The Collection Manager continues to call your callback function, requesting more of the flattened stream of bytes each time, until it has unflattened the entire collection. Your callback function can read the flattened data from any source you choose: it could read the flattened data from a handle-based block of memory, it could read the flattened data from disk, and so on.

To create a flattened collection that includes only those collection items whose attributes match a specified pattern, use the `FlattenPartialCollection` (page 21) function.

To create a flattened collection that includes every item in a collection, use the `FlattenCollection` (page 19) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

UnflattenCollectionFromHdl

Unflattens a collection that was flattened using the `FlattenCollectionToHdl` utility function.

```
OSErr UnflattenCollectionFromHdl (
   Collection aCollection,
   Handle flattened
);
```

Parameters

aCollection

A reference to a collection object in which to store the unflattened information. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattened

A handle to the data that was previously flattened. You must provide a valid Macintosh Memory Manager handle in this parameter.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
To unflatten a collection directly from disk, use the `UnflattenCollection` (page 45) function.

To flatten a collection to a block of memory referenced by a handle, use the `FlattenCollectionToHdl` (page 20) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

Callbacks

CollectionExceptionProcPtr

Defines a pointer to an error handling callback function that handles errors that occur when operating on a collection object.

```
typedef OSErr (*CollectionExceptionProcPtr) (  Collection c,  OSErr status );
```

If you name your function `MyCollectionExceptionProc`, you would declare it like this:

```
OSErr MyCollectionExceptionProc (
    Collection c,
    OSErr status
);
```

Parameters

c

A reference to the collection object for which the error occurred.

status

The result code associated with the error that occurred.

Return Value
A result code. See "Result Codes" (page 57).

Discussion
You create this function to install in a collection object using the `SetCollectionExceptionProc` (page 43) function. Subsequently, whenever the Collection Manager is operating on that collection object and an error occurs, the Collection Manager calls this function, sending it a reference to the collection for which the error occurred and the result code associated with the error. You can use this information to handle the error appropriately for your application.

You can use an exception procedure to respond to an error in a number of ways:

- You can change the error from one result code to another by returning as the function result the new result code.
- You can handle the error and return the `noErr` error code, which indicates that the Collection Manager should return control to the place in your application that generated the error, as if no error had occurred.
- You can use the ANSI C functions `setjmp` and `longjmp` to jump out of the exception procedure into code to handle the error.

To install an exception procedure in a collection object, use the `SetCollectionExceptionProc` (page 43) function.

To obtain a pointer to an existing exception procedure in a collection object, use the `GetCollectionExceptionProc` (page 23) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Collections.h`

CollectionFlattenProcPtr

Defines a pointer to a flattening callback function that reads or writes flattened collection data.

```
typedef OSErr (*CollectionFlattenProcPtr)
(
    SInt32 size,
    void * data,
    void * refCon
);
```

If you name your function `MyCollectionFlattenProc`, you would declare it like this:

```
OSErr MyCollectionFlattenProc (
    SInt32 size,
    void * data,
    void * refCon
);
```

Parameters

size

The size of the block of flattened data to read or write.Your function should read or copy the requested number of bytes of flattened data into the block of memory pointed to by the `data` parameter.

data

A pointer to the block of flattened data. When flattening, this pointer points to the data your callback function should write. When unflattening, your callback function should read flattened data into the memory pointed to by this parameter.

refCon

A value you provide to the `FlattenCollection` function or `UnflattenCollection` function that the Collection Manager passes on to your callback function. You can use this parameter as a pointer to a structure containing relevant state information you need when reading or writing the flattened data.

Return Value

A result code. See "Result Codes" (page 57). If the execution of this function results in any fatal error, you should return the error code back to the Collection Manager as the function result. If the function executes successfully, you should return the `noErr` error code as the function result.

Discussion

You create this function to pass to the `FlattenCollection` (page 19), `FlattenPartialCollection` (page 21), and `UnflattenCollection` (page 45) functions when flattening or unflattening a collection.

As the Collection Manager is flattening a collection, it repeatedly calls this callback function to process sequential blocks of flattened data. Each time it calls this function, it provides a pointer to the current block of flattened data in the `data` parameter and the size of the current block in the `size` parameter. You can process this data in a number of ways: appending it to a handle-based block of memory, writing it to disk, and so on.

When unflattening a collection, the Collection Manager repeatedly calls this function to obtain blocks of flattened data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Collections.h`

Data Types

Collection

Defines defines a reference to an opaque type that your compiler can type-check.

```
typedef struct OpaqueCollection * Collection;
```

Discussion

The Collection Manager provides you with access to a collection object through a `Collection` reference. The `Collection` type defines a reference type that your compiler can type-check; it does not define a pointer to a publicly defined data structure. The contents of the collection object are private; you must use the Collection Manager functions to manipulate collection objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Collections.h`

CollectionExceptionUPP

Defines a universal procedure pointer to an error-handling callback.

```
typedef CollectionExceptionProcPtr CollectionExceptionUPP;
```

Discussion
For more information, see the description of the `CollectionExceptionProcPtr` (page 47) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Collections.h`

CollectionFlattenUPP

Defines a universal procedure pointer to a data-flattening callback.

```
typedef CollectionFlattenProcPtr CollectionFlattenUPP;
```

Discussion
For more information, see the description of the `CollectionFlattenProcPtr` (page 48)callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Collections.h`

CollectionTag

Defines a data type for a collection tag.

```
typedef FourCharCode CollectionTag;
```

Discussion
Each item in a collection is uniquely identified by its collection tag and its collection ID. The collection tag is a four-character identifier, similar to the identifiers used for resources.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Collections.h`

Constants

Attribute Bit Masks

Used to test or set a particular collection item attribute.

```
enum {
    kCollectionUser0Mask = 1L << kCollectionUser0Bit,
    kCollectionUser1Mask = 1L << kCollectionUser1Bit,
    kCollectionUser2Mask = 1L << kCollectionUser2Bit,
    kCollectionUser3Mask = 1L << kCollectionUser3Bit,
    kCollectionUser4Mask = 1L << kCollectionUser4Bit,
    kCollectionUser5Mask = 1L << kCollectionUser5Bit,
    kCollectionUser6Mask = 1L << kCollectionUser6Bit,
    kCollectionUser7Mask = 1L << kCollectionUser7Bit,
    kCollectionUser8Mask = 1L << kCollectionUser8Bit,
    kCollectionUser9Mask = 1L << kCollectionUser9Bit,
    kCollectionUser10Mask = 1L << kCollectionUser10Bit,
    kCollectionUser11Mask = 1L << kCollectionUser11Bit,
    kCollectionUser12Mask = 1L << kCollectionUser12Bit,
    kCollectionUser13Mask = 1L << kCollectionUser13Bit,
    kCollectionUser14Mask = 1L << kCollectionUser14Bit,
    kCollectionUser15Mask = 1L << kCollectionUser15Bit,
    kCollectionReserved0Mask = 1L << kCollectionReserved0Bit,
    kCollectionReserved1Mask = 1L << kCollectionReserved1Bit,
    kCollectionReserved2Mask = 1L << kCollectionReserved2Bit,
    kCollectionReserved3Mask = 1L << kCollectionReserved3Bit,
    kCollectionReserved4Mask = 1L << kCollectionReserved4Bit,
    kCollectionReserved5Mask = 1L << kCollectionReserved5Bit,
    kCollectionReserved6Mask = 1L << kCollectionReserved6Bit,
    kCollectionReserved7Mask = 1L << kCollectionReserved7Bit,
    kCollectionReserved8Mask = 1L << kCollectionReserved8Bit,
    kCollectionReserved9Mask = 1L << kCollectionReserved9Bit,
    kCollectionReserved10Mask = 1L << kCollectionReserved10Bit,
    kCollectionReserved11Mask = 1L << kCollectionReserved11Bit,
    kCollectionReserved12Mask = 1L << kCollectionReserved12Bit,
    kCollectionReserved13Mask = 1L << kCollectionReserved13Bit,
    kCollectionPersistenceMask = 1L << kCollectionPersistenceBit,
    kCollectionLockMask = 1L << kCollectionLockBit
};
```

Discussion

Using the attribute bit numbers, the Collection Manager provides convenient attribute masks for each of the attributes. You can use these attribute masks when testing or setting a particular collection item attribute.

Attribute Bit Masks (Old)

Used to test or set a particular collection item attribute.

```
enum {
    collectionUser0Mask = kCollectionUser0Mask,
    collectionUser1Mask = kCollectionUser1Mask,
    collectionUser2Mask = kCollectionUser2Mask,
    collectionUser3Mask = kCollectionUser3Mask,
    collectionUser4Mask = kCollectionUser4Mask,
    collectionUser5Mask = kCollectionUser5Mask,
    collectionUser6Mask = kCollectionUser6Mask,
    collectionUser7Mask = kCollectionUser7Mask,
    collectionUser8Mask = kCollectionUser8Mask,
    collectionUser9Mask = kCollectionUser9Mask,
    collectionUser10Mask = kCollectionUser10Mask,
    collectionUser11Mask = kCollectionUser11Mask,
    collectionUser12Mask = kCollectionUser12Mask,
    collectionUser13Mask = kCollectionUser13Mask,
    collectionUser14Mask = kCollectionUser14Mask,
    collectionUser15Mask = kCollectionUser15Mask,
    collectionReserved0Mask = kCollectionReserved0Mask,
    collectionReserved1Mask = kCollectionReserved1Mask,
    collectionReserved2Mask = kCollectionReserved2Mask,
    collectionReserved3Mask = kCollectionReserved3Mask,
    collectionReserved4Mask = kCollectionReserved4Mask,
    collectionReserved5Mask = kCollectionReserved5Mask,
    collectionReserved6Mask = kCollectionReserved6Mask,
    collectionReserved7Mask = kCollectionReserved7Mask,
    collectionReserved8Mask = kCollectionReserved8Mask,
    collectionReserved9Mask = kCollectionReserved9Mask,
    collectionReserved10Mask = kCollectionReserved10Mask,
    collectionReserved11Mask = kCollectionReserved11Mask,
    collectionReserved12Mask = kCollectionReserved12Mask,
    collectionReserved13Mask = kCollectionReserved13Mask,
    collectionPersistenceMask = kCollectionPersistenceMask,
    collectionLockMask = kCollectionLockMask
};
```

Discussion
Using the attribute bit numbers, the Collection Manager provides convenient attribute masks for each of the attributes. You can use these attribute masks when testing or setting a particular collection item attribute.

Attribute Bit Numbers

Provides constant names for each of the bits in a collection item attributes.

```
enum {
    kCollectionUser0Bit = 0,
    kCollectionUser1Bit = 1,
    kCollectionUser2Bit = 2,
    kCollectionUser3Bit = 3,
    kCollectionUser4Bit = 4,
    kCollectionUser5Bit = 5,
    kCollectionUser6Bit = 6,
    kCollectionUser7Bit = 7,
    kCollectionUser8Bit = 8,
    kCollectionUser9Bit = 9,
    kCollectionUser10Bit = 10,
    kCollectionUser11Bit = 11,
    kCollectionUser12Bit = 12,
    kCollectionUser13Bit = 13,
    kCollectionUser14Bit = 14,
    kCollectionUser15Bit = 15,
    kCollectionReserved0Bit = 16,
    kCollectionReserved1Bit = 17,
    kCollectionReserved2Bit = 18,
    kCollectionReserved3Bit = 19,
    kCollectionReserved4Bit = 20,
    kCollectionReserved5Bit = 21,
    kCollectionReserved6Bit = 22,
    kCollectionReserved7Bit = 23,
    kCollectionReserved8Bit = 24,
    kCollectionReserved9Bit = 25,
    kCollectionReserved10Bit = 26,
    kCollectionReserved11Bit = 27,
    kCollectionReserved12Bit = 28,
    kCollectionReserved13Bit = 29,
    kCollectionPersistenceBit = 30,
    kCollectionLockBit = 31
};
```

Discussion
The Collection Manager provides the attribute bit numbers enumeration to provide constant names for each of the bits in a collection item's attributes.

The lower 16 bits of the attributes property of a collection item represent the user-defined attributes. You can use these attributes for any purpose suitable to your application.

The upper 16 bits are reserved for use by Apple Computer, Inc. Currently, the 2 high bits are defined: bit 30 represents the persistence attribute and bit 31 represents the lock attribute.

Attribute Bit Numbers (Old)

Provides constant names for each of the bits in a collection item attributes.

```
enum {
    collectionUser0Bit = kCollectionUser0Bit,
    collectionUser1Bit = kCollectionUser1Bit,
    collectionUser2Bit = kCollectionUser2Bit,
    collectionUser3Bit = kCollectionUser3Bit,
    collectionUser4Bit = kCollectionUser4Bit,
    collectionUser5Bit = kCollectionUser5Bit,
    collectionUser6Bit = kCollectionUser6Bit,
    collectionUser7Bit = kCollectionUser7Bit,
    collectionUser8Bit = kCollectionUser8Bit,
    collectionUser9Bit = kCollectionUser9Bit,
    collectionUser10Bit = kCollectionUser10Bit,
    collectionUser11Bit = kCollectionUser11Bit,
    collectionUser12Bit = kCollectionUser12Bit,
    collectionUser13Bit = kCollectionUser13Bit,
    collectionUser14Bit = kCollectionUser14Bit,
    collectionUser15Bit = kCollectionUser15Bit,
    collectionReserved0Bit = kCollectionReserved0Bit,
    collectionReserved1Bit = kCollectionReserved1Bit,
    collectionReserved2Bit = kCollectionReserved2Bit,
    collectionReserved3Bit = kCollectionReserved3Bit,
    collectionReserved4Bit = kCollectionReserved4Bit,
    collectionReserved5Bit = kCollectionReserved5Bit,
    collectionReserved6Bit = kCollectionReserved6Bit,
    collectionReserved7Bit = kCollectionReserved7Bit,
    collectionReserved8Bit = kCollectionReserved8Bit,
    collectionReserved9Bit = kCollectionReserved9Bit,
    collectionReserved10Bit = kCollectionReserved10Bit,
    collectionReserved11Bit = kCollectionReserved11Bit,
    collectionReserved12Bit = kCollectionReserved12Bit,
    collectionReserved13Bit = kCollectionReserved13Bit,
    collectionPersistenceBit = kCollectionPersistenceBit,
    collectionLockBit = kCollectionLockBit
};
```

Discussion
The Collection Manager provides the attribute bit numbers enumeration to provide constant names for each of the bits in a collection item's attributes.

The lower 16 bits of the attributes property of a collection item represent the user-defined attributes. You can use these attributes for any purpose suitable to your application.

The upper 16 bits are reserved for use by Apple Computer, Inc. Currently, the 2 high bits are defined: bit 30 represents the persistence attribute and bit 31 represents the lock attribute.

Attributes Masks

Used to specify attributes for any of the attribute-related Collection Manager functions.

```
enum {
    kCollectionNoAttributes = 0x00000000,
    kCollectionAllAttributes = 0xFFFFFFFF,
    kCollectionUserAttributes = 0x0000FFFF,
    kCollectionDefaultAttributes = 0x40000000
};
```

Constants

`kCollectionNoAttributes`

Specifies a mask in which all collection attributes are clear. You might use this constant when clearing all the attributes of an item or when testing whether all of an item's attributes are clear.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`kCollectionAllAttributes`

Specifies a mask in which all collection attributes are set. You might use this constant as a mask to indicate that you want to edit or test every attribute of an item, or you might use it to set every attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`kCollectionUserAttributes`

Specifies a mask in which the user attributes are set and the reserved attributes are clear. You might use this constant as a mask to indicate that you want to edit or test only the user attributes of an item, or you might use it to set every user attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`kCollectionDefaultAttributes`

Specifies a mask in which the persistent attribute is set and all other attributes are clear. You might use this constant when testing to see if an item's attributes have been edited.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

Discussion

The Collection Manager provides four convenient attributes masks that you can use when specifying attributes for any of the attribute-related Collection Manager functions. You can also use the attribute bit masks as masks for individual attributes.

Attributes Masks (Old)

Used to specify attributes for any of the attribute-related Collection Manager functions.

```
enum {
    noCollectionAttributes = kCollectionNoAttributes,
    allCollectionAttributes = kCollectionAllAttributes,
    userCollectionAttributes = kCollectionUserAttributes,
    defaultCollectionAttributes = kCollectionDefaultAttributes
};
```

Constants

`noCollectionAttributes`

Specifies a mask in which all collection attributes are clear. You might use this constant when clearing all the attributes of an item or when testing whether all of an item's attributes are clear.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`allCollectionAttributes`

Specifies a mask in which all collection attributes are set. You might use this constant as a mask to indicate that you want to edit or test every attribute of an item, or you might use it to set every attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`userCollectionAttributes`

Specifies a mask in which the user attributes are set and the reserved attributes are clear. You might use this constant as a mask to indicate that you want to edit or test only the user attributes of an item, or you might use it to set every user attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`defaultCollectionAttributes`

Specifies a mask in which the persistent attribute is set and all other attributes are clear. You might use this constant when testing to see if an item's attributes have been edited.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

Discussion

The Collection Manager provides four convenient attributes masks that you can use when specifying attributes for any of the attribute-related Collection Manager functions. You can also use the attribute bit masks as masks for individual attributes.

Optional Return Value Constants

Used to specify that you do not want a particular piece of information.

```
enum {
    kCollectionDontWantTag = 0,
    kCollectionDontWantId = 0,
    kCollectionDontWantSize = 0,
    kCollectionDontWantAttributes = 0,
    kCollectionDontWantIndex = 0,
    kCollectionDontWantData = 0
};
```

Discussion
Many of the Collection Manager functions return multiple pieces of information. For most of these functions, you can specify that you do not want a specific piece of information to be returned by specifying `NULL` for the corresponding parameter when calling the function.

The Collection Manager provides the optional return value constants to make your code easier to read when specifying that you are not interested in obtaining certain types of information. You can use these enumeration constants in place of the more generic constant `NULL` when specifying that you do not want to receive certain optional return values from a function.

Optional Return Value Constants (Old)

Used to specify that you do not want a particular piece of information.

```
enum {
    dontWantTag = kCollectionDontWantTag,
    dontWantId = kCollectionDontWantId,
    dontWantSize = kCollectionDontWantSize,
    dontWantAttributes = kCollectionDontWantAttributes,
    dontWantIndex = kCollectionDontWantIndex,
    dontWantData = kCollectionDontWantData
};
```

Discussion
Many of the Collection Manager functions return multiple pieces of information. For most of these functions, you can specify that you do not want a specific piece of information to be returned by specifying `NULL` for the corresponding parameter when calling the function.

The Collection Manager provides the optional return value constants to make your code easier to read when specifying that you are not interested in obtaining certain types of information. You can use these enumeration constants in place of the more generic constant `NULL` when specifying that you do not want to receive certain optional return values from a function.

Result Codes

The most common result codes returned by the Collection Manager are listed in the table below.

Result Code	Value	Description
`collectionItemLockedErr`	-5750	Available in Mac OS X v10.0 and later.
`collectionItemNotFoundErr`	-5751	Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`collectionIndexRangeErr`	-5752	Available in Mac OS X v10.0 and later.
`collectionVersionErr`	-5753	Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *Collection Manager Reference*.

Date	Notes
2003-04-01	Added documentation for functions that work with universal procedure pointers.
	Added abstracts for data types and constants.
2003-02-01	Updated formatting.
	Grouped new functions
2001-07-01	Last version of this document.

Document Revision History

Index

N

O

P

R

S

U