CFTree Reference

Core Foundation

2005-12-06

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

Simultaneously published in the United States and Canada.

Contents

CFTree Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Companion guide	Collections Programming Topics for Core Foundation
Declared in	CFTree.h

Overview

You use CFTree to create tree structures that represent hierarchical organizations of information. In such structures, each tree node has exactly one parent tree (except for the root tree, which has no parent) and can have multiple children. Each CFTree object in the structure has a context associated with it; this context includes some program-defined data as well as callbacks that operate on that data. The program-defined data is often used as the basis for determining where CFTree objects fit within the structure. All CFTree objects are mutable.

You create a CFTree object using the `CFTreeCreate` (page 8) function. This function takes an allocator and pointer to a `CFTreeGetContext` (page 10) structure as parameters. The `CFTreeContext` (page 18) structure contains the program-defined data and callbacks needed to describe, retain, and release that data. If you do not implement these callbacks, your program-defined data will not be retained or released when trees are added and removed from a parent.

Each CFTree object has a parent and list of children, all of which may be `NULL`. CFTree provides functions for adding and removing tree objects from the tree structure. Use the `CFTreeAppendChild` (page 7), `CFTreeInsertSibling` (page 12), or `CFTreePrependChild` (page 13) functions to add trees to a tree structure, and the `CFTreeRemove` (page 13) or `CFTreeRemoveAllChildren` (page 14) functions to remove trees.

For the purposes of memory management, CFTree can be thought of as a collection. Typically the only object that retains a child tree is its parent. Usually, therefore, when you remove a child tree from a tree, the child tree is destroyed. If you want to use a child tree after you remove it from its parent, you should retain the child tree first, prior to removing it.

Releasing a tree releases its child trees, and all of their child trees (recursively). Note also that the final release of a tree (when its retain count decreases to zero) causes all of its child trees, and all of their child trees (recursively), to be destroyed, regardless of their retain counts. Releasing a child that is still in a tree is therefore a programming error, and may cause your application to crash.

You can use any of the get functions (functions containing the word “Get”) to obtain the parent, children, or attributes of a tree. For example, use `CFTreeGetChildAtIndex` (page 9) to obtain a child of a tree at a specified location. In common with other Core Foundation “Get” functions, these functions do not retain the tree that is returned. If you are making other modifications to the tree, you should either retain or make a deep copy of the child tree returned.

You can apply a function to all children of a tree using the `CFTreeApplyFunctionToChildren` (page 8) function, and sort children of a tree using the `CFTreeSortChildren` (page 15) function.

Functions by Task

Creating Trees

`CFTreeCreate` (page 8)
Creates a new CFTree object.

Modifying a Tree

`CFTreeAppendChild` (page 7)
Adds a new child to a tree as the last in its list of children.

`CFTreeInsertSibling` (page 12)
Inserts a new sibling after a given tree.

`CFTreeRemoveAllChildren` (page 14)
Removes all the children of a tree.

`CFTreePrependChild` (page 13)
Adds a new child to the specified tree as the first in its list of children.

`CFTreeRemove` (page 13)
Removes a tree from its parent.

`CFTreeSetContext` (page 14)
Replaces the context of a tree by releasing the old information pointer and retaining the new one.

Sorting a Tree

`CFTreeSortChildren` (page 15)
Sorts the immediate children of a tree using a specified comparator function.

Examining a Tree

`CFTreeFindRoot` (page 9)
Returns the root tree of a given tree.

`CFTreeGetChildAtIndex` (page 9)
Returns the child of a tree at the specified index.

`CFTreeGetChildCount` (page 10)
Returns the number of children in a tree.

`CFTreeGetChildren` (page 10)
Fills a buffer with children from the tree.

`CFTreeGetContext` (page 10)
Returns the context of the specified tree.

`CFTreeGetFirstChild` (page 11)
Returns the first child of a tree.

`CFTreeGetNextSibling` (page 11)
Returns the next sibling, adjacent to a given tree, in the parent's children list.

`CFTreeGetParent` (page 12)
Returns the parent of a given tree.

Performing an Operation on Tree Elements

`CFTreeApplyFunctionToChildren` (page 8)
Calls a function once for each immediate child of a tree.

Getting the Tree Type ID

`CFTreeGetTypeID` (page 12)
Returns the type identifier of the CFTree opaque type.

Functions

CFTreeAppendChild

Adds a new child to a tree as the last in its list of children.

```
void CFTreeAppendChild (
   CFTreeRef tree,
   CFTreeRef newChild
);
```

Parameters

tree
The tree to which to add *newChild*.

newChild
The child tree to add to *tree*. If this parameter is a tree which is already a child of any other tree, the behavior is undefined.

Discussion
When a child tree is added to another tree, the child tree is retained by its new parent.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeApplyFunctionToChildren

Calls a function once for each immediate child of a tree.

```
void CFTreeApplyFunctionToChildren (
   CFTreeRef tree,
   CFTreeApplierFunction applier,
   void *context
);
```

Parameters

tree

The tree to operate upon.

applier

The callback function to call once for each child in *tree*. The function must be able to apply to all the values in the tree.

context

A pointer-sized program-defined value that is passed to the applier function, but is otherwise unused by this function.

Discussion

Note that the applier only operates one level deep—it does not operate on descendants further removed than the immediate children of a tree. If the tree is mutable, it is unsafe for the applied function to change the contents of the tree.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeCreate

Creates a new CFTree object.

```
CFTreeRef CFTreeCreate (
   CFAllocatorRef allocator,
   const CFTreeContext *context
);
```

Parameters

allocator

The allocator to use to allocate memory for the new tree. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

context

The `CFTreeContext` (page 18) structure to be copied and used as the context of the new tree. The information pointer will be retained by the tree if a retain function is provided. If this value is not a valid C pointer to a `CFTreeContext` structure-sized block of storage, the result is undefined. If the version number of the storage is not a valid `CFTreeContext` version number, the result is undefined.

Return Value

A new CFTree object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeFindRoot

Returns the root tree of a given tree.

```
CFTreeRef CFTreeFindRoot (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value
The root of *tree* where root is defined as a tree without a parent. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeGetChildAtIndex

Returns the child of a tree at the specified index.

```
CFTreeRef CFTreeGetChildAtIndex (
   CFTreeRef tree,
   CFIndex idx
);
```

Parameters

tree

The tree to examine.

idx

The index of the child obtain. The value must be less than the number of children in *tree*.

Return Value
The child tree at *idx*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeGetChildCount

Returns the number of children in a tree.

```
CFIndex CFTreeGetChildCount (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value

The number of children in *tree*.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreeGetChildren

Fills a buffer with children from the tree.

```
void CFTreeGetChildren (
   CFTreeRef tree,
   CFTreeRef *children
);
```

Parameters

tree

The tree to examine.

children

The C array of pointer-sized values to be filled with the children from *tree*. This value must be a valid pointer to a C array of at least the size of the number of children in *tree*. Use the `CFTreeGetChildCount` (page 10) function to obtain the number of children in *tree*. You are responsible for retaining and releasing the returned objects as needed.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreeGetContext

Returns the context of the specified tree.

```
void CFTreeGetContext (
   CFTreeRef tree,
   CFTreeContext *context
);
```

Parameters

tree

The tree to examine.

context

The `CFTreeContext` (page 18) structure to be filled in with the context of the specified tree. This value must be a valid C pointer to a `CFTreeContext` structure-sized block of storage. If the version number of the storage is not a valid `CFTreeContext` structure version number, the result is undefined.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeGetFirstChild

Returns the first child of a tree.

```
CFTreeRef CFTreeGetFirstChild (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value

The first child of *tree*. Ownership follows the Get Rule.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeGetNextSibling

Returns the next sibling, adjacent to a given tree, in the parent's children list.

```
CFTreeRef CFTreeGetNextSibling (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value
The next sibling, adjacent to *tree*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeGetParent

Returns the parent of a given tree.

```
CFTreeRef CFTreeGetParent (
   CFTreeRef tree
);
```

Parameters

tree
The tree to examine.

Return Value
The parent of *tree*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeGetTypeID

Returns the type identifier of the CFTree opaque type.

```
CFTypeID CFTreeGetTypeID (
   void
);
```

Return Value
The type identifier of the CFTree opaque type.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeInsertSibling

Inserts a new sibling after a given tree.

```
void CFTreeInsertSibling (
   CFTreeRef tree,
   CFTreeRef newSibling
);
```

Parameters

tree

The tree after which to insert *newSibling*. *tree* must have a parent.

newSibling

The sibling to add. *newSibling* must not have a parent.

Discussion

When a child tree is added to another tree, the child tree is retained by its new parent.

If you want to manipulate an existing tree structure, since *newSibling* must not have a parent you need to remove a tree from its parent in order to move it to a new position. If you do this, you should retain the tree before you actually remove it from its parent (see CFTreeRemove (page 13)).

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreePrependChild

Adds a new child to the specified tree as the first in its list of children.

```
void CFTreePrependChild (
   CFTreeRef tree,
   CFTreeRef newChild
);
```

Parameters

tree

The tree to which to add *newChild*.

newChild

The child tree to add to *tree*. This value must not be a child of another tree.

Discussion

When a child tree is added to another tree, the child tree is retained by its new parent.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreeRemove

Removes a tree from its parent.

```
void CFTreeRemove (
   CFTreeRef tree
);
```

Parameters

tree

The tree to remove from its parent.

Discussion

When a child tree is removed from its parent, the parent releases it. If you want to use the child after you have removed it, you should ensure you retain it before removing it from its parent.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeRemoveAllChildren

Removes all the children of a tree.

```
void CFTreeRemoveAllChildren (
   CFTreeRef tree
);
```

Parameters

tree

The tree to modify.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeSetContext

Replaces the context of a tree by releasing the old information pointer and retaining the new one.

```
void CFTreeSetContext (
   CFTreeRef tree,
   const CFTreeContext *context
);
```

Parameters

tree

The tree to modify.

context

The CFTreeContext (page 18) structure to be copied and used as the context of the new tree. The information pointer will be retained by the tree if a retain function is provided. If this value is not a valid C pointer to a CFTreeContext structure-sized block of storage, the result is undefined. If the version number of the storage is not a valid CFTreeContext version number, the result is undefined.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeSortChildren

Sorts the immediate children of a tree using a specified comparator function.

```
void CFTreeSortChildren (
   CFTreeRef tree,
   CFComparatorFunction comparator,
   void *context
);
```

Parameters

tree

The tree to sort.

comparator

The function with a comparator function type signature which is used in the sort operation to compare children of the tree. The children of the tree are sorted from least to greatest according to this function.

context

A pointer-sized program-defined value that is passed to the comparator function, but is otherwise unused by this function.

Discussion
Note that the comparator only operates one level deep and does not operate on descendants further removed than the immediate children of a tree node.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

Callbacks

CFTreeApplierFunction

Type of the callback function used by the CFTree apply function.

```
typedef void (*CFTreeApplierFunction) (
   const void *value,
   void *context
);
```

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

```
void MyCallBack (
   const void *value,
   void *context
);
```

Parameters

value

The current child of a tree that is being iterated.

context

The program-defined context parameter that was passed to the applier function.

Discussion

This callback is used by the `CFTreeApplyFunctionToChildren` (page 8) applier function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeCopyDescriptionCallBack

Callback function used to provide a description of the program-defined information pointer.

```
typedef CFStringRef (*CFTreeCopyDescriptionCallBack) (
   const void *info
);
```

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

```
CFStringRef MyCallBack (
   const void *info
);
```

Parameters

info

The program-supplied information pointer provided in a `CFTreeContext` (page 18) structure.

Return Value

A textual description of *info*. The caller is responsible for releasing this object.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeReleaseCallBack

Callback function used to release a previously retained program-defined information pointer.

```
typedef void (*CFTreeReleaseCallBack) (
   const void *info
);
```

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

```
void MyCallBack (
   const void *info
);
```

Parameters

info

The program-supplied information pointer provided in a `CFTreeContext` (page 18) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeRetainCallBack

Callback function used to retain a program-defined information pointer.

```
typedef const void *(*CFTreeRetainCallBack) (
const void *info
);
```

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

```
const void *MyCallBack (
   const void *info
);
```

Parameters

info

The program-supplied information pointer provided in a `CFTreeContext` (page 18) structure.

Return Value

The value to use whenever the information pointer is retained, which is usually the *info* parameter passed to this callback, but may be a different value if a different value should be used.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

Data Types

CFTreeContext

Structure containing program-defined data and callbacks for a CFTree object.

```
struct CFTreeContext {
    CFIndex version;
    void *info;
    CFTreeRetainCallBack retain;
    CFTreeReleaseCallBack release;
    CFTreeCopyDescriptionCallBack copyDescription;
};
typedef struct CFTreeContext CFTreeContext;
```

Fields

`version`
: The version number of the structure type being passed in as a parameter to a CFTree creation function. This structure is version `0`.

`info`
: A C pointer to a program-defined block of data, referred to as the information pointer.

`retain`
: The callback used to retain the `info` field. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. This value may be `NULL`.

`release`
: The callback used to release a previously retained `info` field. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. This value may be `NULL`.

`copyDescription`
: The callback used to provide a description of the `info` field.

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

Declared In
`CFTree.h`

CFTreeRef

A reference to a CFTree object.

```
typedef struct __CFTree *CFTreeRef;
```

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

Declared In
`CFTree.h`

Document Revision History

This table describes the changes to *CFTree Reference*.

Date	Notes
2005-12-06	Made minor changes to text to conform to reference consistency guidelines.
2005-04-29	Moved Introduction to new Introduction page. Clarified memory management issues in Introduction and tree manipulation function descriptions.
2004-12-02	Added note to Introduction regarding removal of children from tree.
2004-02-11	Correction to parameter description of `CFTreeGetChildAtIndex`.
2003-01-01	First version of this document.

Document Revision History

Index

C