Component Manager Reference

Carbon > Runtime Architecture

2006-07-17

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

Simultaneously published in the United States and Canada.

Contents

Component Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Components.h

Overview

You can use the Component Manager to allow your application to find and utilize various software objects (components) at run time. You can create your own components, and you can use the Component Manager to help manage your components. A component is a piece of code that provides a defined set of services to one or more clients. Applications, system extensions, as well as other components can use the services of a component. A component typically provides a specific type of service to its clients. For example, a component might provide image compression or image decompression capabilities; an application could call such a component, providing the image to compress, and the component could perform the desired operation and return the compressed image to the application. The Component Manager provides access to components and manages them by, for example, keeping track of the currently available components and routing requests to the appropriate component.

Functions by Task

Finding Components

`CountComponents` (page 20)
Returns the number of registered components that meet the selection criteria specified by your application.

`FindNextComponent` (page 24)
Returns the component identifier for the next registered component that meets the selection criteria specified by your application.

`GetComponentListModSeed` (page 28)
Allows your application to determine if the list of registered components has changed.

`GetComponentTypeModSeed` (page 31)

`ResolveComponentAlias` (page 42)

Opening and Closing Components

`OpenAComponent` (page 35)

`OpenADefaultComponent` (page 36)

`OpenComponent` (page 36)
Opens a connection to the component with the component identifier specified by your application.

`OpenDefaultComponent` (page 38)
Opens a connection to a registered component of the component type and subtype specified by your application.

`CloseComponent` (page 19)
Terminates your application's connection to a component.

Getting Information About Components

`GetComponentIconSuite` (page 25)
Returns a handle to a component's icon suite to your application.

`GetComponentInfo` (page 26)
Returns to your application the registration information for a component.

`GetComponentPublicIndString` (page 29)

`GetComponentPublicResource` (page 29)

`GetComponentPublicResourceList` (page 30)

`ComponentFunctionImplemented` (page 71) Deprecated in Mac OS X v10.5
Allows your application to determine whether a component supports a specified request.

`GetComponentVersion` (page 72) Deprecated in Mac OS X v10.5
Returns the version number of a component to your application.

Retrieving Component Errors

`GetComponentInstanceError` (page 27)
Returns to your application the last error generated by a specific connection to a component.

Calling Component Functions

`CallComponentOpen` (page 16)

`CallComponentClose` (page 12)

CallComponentCanDo (page 12)

CallComponentVersion (page 18)

CallComponentRegister (page 16)

CallComponentTarget (page 17)

CallComponentUnregister (page 17)

CallComponentDispatch (page 13)

CallComponentGetMPWorkFunction (page 15)

CallComponentGetPublicResource (page 16)

Accessing the Thread Safety Mode

CSSetComponentsThreadMode (page 21)
: Sets whether or not using thread-unsafe components is allowed in the current thread.

CSGetComponentsThreadMode (page 21)
: Indicates whether using thread-unsafe components is allowed in the current thread.

Creating and Managing Universal Procedure Pointers

NewComponentRoutineUPP (page 34)
: Creates a new universal procedure pointer (UPP) to a component routine callback function.

InvokeComponentRoutineUPP (page 32)
: Calls your component routine callback function

DisposeComponentRoutineUPP (page 23)
: Disposes of the universal procedure pointer (UPP) to a component routine callback function.

NewComponentFunctionUPP (page 33)

DisposeComponentFunctionUPP (page 23)

NewComponentMPWorkFunctionUPP (page 34)

InvokeComponentMPWorkFunctionUPP (page 32)

DisposeComponentMPWorkFunctionUPP (page 23)

NewGetMissingComponentResourceUPP (page 34)

InvokeGetMissingComponentResourceUPP (page 33)

DisposeGetMissingComponentResourceUPP (page 24)

Registering Components

RegisterComponent (page 38)
Registers a component stored in memory.

RegisterComponentResource (page 41)
Registers a component stored in a resource file.

RegisterComponentResourceFile (page 42)
Registers all component resources in the given resource file.

UnregisterComponent (page 46)
Removes a component from the Component Manager's registration list.

RegisterComponentFileRef (page 40)

RegisterComponentFileRefEntries (page 40)

RegisterComponentFile (page 73) Deprecated in Mac OS X v10.5

RegisterComponentFileEntries (page 73) Deprecated in Mac OS X v10.5

Dispatching to Component Functions

CallComponentFunction (page 13)
Invokes the specified function of your component.

CallComponentFunctionWithStorage (page 14)
Invokes the specified function of your component.

CallComponentFunctionWithStorageProcInfo (page 15)

Managing Component Connections

CountComponentInstances (page 20)
Determines the number of open connections being managed by a specified component.

GetComponentInstanceStorage (page 28)
Allows your component to retrieve a handle to the memory associated with a connection.

SetComponentInstanceStorage (page 43)
Allows your component to associate memory with a connection.

`ComponentSetTarget` (page 71) Deprecated in Mac OS X v10.5
Calls a component's target request function and informs a component that it has been targeted by another component.

Setting Component Errors

`SetComponentInstanceError` (page 43)
Passes error information to the Component Manager which sets the current error value for the appropriate connection.

Working With Component Reference Constants

`GetComponentRefcon` (page 30)
Retrieves the value of the reference constant for your component.

`SetComponentRefcon` (page 44)
Sets the reference constant for your component.

Accessing a Component's Resource File

`OpenAComponentResFile` (page 35)

`OpenComponentResFile` (page 37)
Allows your component to gain access to its resource file.

`CloseComponentResFile` (page 19)
Closes the resource file that your component opened previously with the `OpenComponentResFile` function.

`GetComponentResource` (page 31)

`GetComponentIndString` (page 26)

Calling Other Components

`DelegateComponentCall` (page 22)
Allows your component to pass on a request to a specified component.

Capturing Components

`CaptureComponent` (page 18)
Allows your component to capture another component.

`UncaptureComponent` (page 45)
Allows your component to uncapture a previously captured component.

Changing the Default Search Order

`SetDefaultComponent` (page 45)
Changes the search order for registered components.

Functions

CallComponentCanDo

```
ComponentResult CallComponentCanDo (
   ComponentInstance ci,
   SInt16 ftnNumber
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentClose

```
ComponentResult CallComponentClose (
   ComponentInstance ci,
   ComponentInstance self
);
```

Parameters

ci

self

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentDispatch

```
ComponentResult CallComponentDispatch (
   ComponentParameters *cp
);
```

Parameters

cp

Return Value

See the description of the `ComponentResult` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

CallComponentFunction

Invokes the specified function of your component.

```
ComponentResult CallComponentFunction (
   ComponentParameters *params,
   ComponentFunctionUPP func
);
```

Parameters

params

A pointer to the `ComponentDescription` (page 50) structure that your component received from the Component Manager. These are the parameters originally provided by the application that called your component.

func

A universal procedure pointer to the component function that is to handle the request. The Component Manager calls the function referred to by the `func` parameter, using Pascal calling conventions, with the parameters that were originally provided by the application that called your component. The function referred to by this parameter must return a function result of type `ComponentResult` indicating the success or failure of the operation. See the `ComponentRoutineProcPtr` (page 47) callback for more information on component functions.

Return Value

The value that is returned by the function referred to by the `func` parameter. Your component should use this value to set the current error for this connection. You can use the `SetComponentInstanceError` (page 43) function to set the current error.

Discussion

When an application requests service from your component, your component receives a component parameters structure containing the parameters that the application provided when it called your component. Your component can use this structure to access the parameters directly. Alternatively, you can use either this function or `CallComponentFunctionWithStorage` (page 14) to extract those parameters and pass them to a subroutine of your component. By taking advantage of these functions, you can simplify the structure of your component code.

If your component subroutine does not need global data, your component should use this function. If your component subroutine requires memory in which to store global data for the component, your component must use `CallComponentFunctionWithStorage`.

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

Declared In
`Components.h`

CallComponentFunctionWithStorage

Invokes the specified function of your component.

```
ComponentResult CallComponentFunctionWithStorage (
   Handle storage,
   ComponentParameters *params,
   ComponentFunctionUPP func
);
```

Parameters

storage

A handle to the memory associated with the current connection. The Component Manager provides this handle to your component along with the request.

params

A pointer to the `ComponentParameters` (page 54) structure that your component received from the Component Manager. These are the parameters originally provided by the application that called your component.

func

A universal procedure pointer to the component function that is to handle the request. The Component Manager calls the function referred to by the `func` parameter, using Pascal calling conventions, with the parameters that were originally provided by the application that called your component. These parameters are preceded by a handle to the memory associated with the current connection. The function referred to by the `func` parameter must return a function result of type `ComponentResult` indicating the success or failure of the operation. See the `ComponentRoutineProcPtr` (page 47) callback for more information on component functions.

Return Value
The value that is returned by the function referred to by the `func` parameter. Your component should use this value to set the current error for this connection. Use the `SetComponentInstanceError` (page 43) function to set the current error for a connection.

Discussion
When an application requests service from your component, your component receives a component parameters structure containing the parameters that the application provided when it called your component. Your component can use this structure to access the parameters directly. Alternatively, you can use either the `CallComponentFunction` (page 13) function or this function to extract those parameters and pass them to a subroutine of your component. By taking advantage of these functions, you can simplify the structure of your component code.

If your component subroutine requires a handle to the memory associated with the connection, you must use this function. You allocate the memory for a given connection each time your component is opened. You inform the Component Manager that a connection has memory associated with it by calling the `SetComponentInstanceError` (page 43) function.

Subroutines of a component that don't need global data should use `CallComponentFunction` instead.

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

Declared In
`Components.h`

CallComponentFunctionWithStorageProcInfo

```
ComponentResult CallComponentFunctionWithStorageProcInfo (
   Handle storage,
   ComponentParameters *params,
   ProcPtr func,
   ProcInfoType funcProcInfo
);
```

Parameters

storage

params

funcProcInfo

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

Declared In
`Components.h`

CallComponentGetMPWorkFunction

```
ComponentResult CallComponentGetMPWorkFunction (
   ComponentInstance ci,
   ComponentMPWorkFunctionUPP *workFunction,
   void **refCon
);
```

Parameters

ci

workFunction

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentGetPublicResource

```
ComponentResult CallComponentGetPublicResource (
   ComponentInstance ci,
   OSType resourceType,
   SInt16 resourceID,
   Handle *resource
);
```

Parameters

ci

resourceType

resource

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentOpen

```
ComponentResult CallComponentOpen (
   ComponentInstance ci,
   ComponentInstance self
);
```

Parameters

ci

self

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentRegister

```
ComponentResult CallComponentRegister (
   ComponentInstance ci
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentTarget

```
ComponentResult CallComponentTarget (
   ComponentInstance ci,
   ComponentInstance target
);
```

Parameters

ci

target

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentUnregister

```
ComponentResult CallComponentUnregister (
   ComponentInstance ci
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentVersion

```
ComponentResult CallComponentVersion (
   ComponentInstance ci
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CaptureComponent

Allows your component to capture another component.

```
Component CaptureComponent (
   Component capturedComponent,
   Component capturingComponent
);
```

Parameters

capturedComponent

The component to be captured. Your component can obtain this identifier from the `FindNextComponent` (page 24) function or from the component registration functions. You can use a component instance here, but you must coerce the data type appropriately.

capturingComponent

Your component. Note that you can use the component instance (appropriately coerced) that your component received in its open request in this parameter.

Return Value
A new component identifier. Your component can use this new identifier to refer to the captured component. For example, your component can open the captured component by providing this identifier to the `OpenComponent` (page 36) structure. Your component must provide this identifier to the `UncaptureComponent` (page 45) function to specify the component to be restored to the search list. If the component you wish to capture is already captured, the component identifier is set to `NULL`. See the description of the `Component` data type.

Discussion
Typically, your component captures another component when you want to override all or some of the features provided by a component or to provide new features. For example, a component called NewMath might capture a component called OldMath. Suppose the NewMath component provides a new function, `DoExponent`. Whenever NewMath gets an exponent request, it can handle the request itself. For all other requests, NewMath might call the OldMath component to perform the request.

After capturing a component, your component might choose to target a particular instance of the captured component.

In response to this function, the Component Manager removes the specified component from the list of available components. As a result, applications cannot retrieve information about the captured component or gain access to it. Current clients of the captured component are not affected by this function.

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

Declared In
`Components.h`

CloseComponent

Terminates your application's connection to a component.

```
OSErr CloseComponent (
   ComponentInstance aComponentInstance
);
```

Parameters

aComponentInstance

The connection you wish to close. Your application obtains the component instance from the `OpenComponent` (page 36) function or the `OpenDefaultComponent` (page 38) function. You can use a component identifier here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Discussion
This function closes only a single connection. If your application has several connections to a single component, you must call it once for each connection.

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

Related Sample Code
WhackedTV

Declared In
`Components.h`

CloseComponentResFile

Closes the resource file that your component opened previously with the `OpenComponentResFile` function.

```
OSErr CloseComponentResFile (
   ResFileRefNum refnum
);
```

Parameters

refnum

The reference number that identifies the resource file to be closed. Your component obtains this value from the `OpenComponentResFile` (page 37) function. Your component must close any open resource files before returning to the calling application.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

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

Declared In
`Components.h`

CountComponentInstances

Determines the number of open connections being managed by a specified component.

```
long CountComponentInstances (
   Component aComponent
);
```

Parameters

aComponent

The component for which you want a count of open connections. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
The number of open connections for the specified component.

Discussion
This function can be useful if you want to restrict the number of connections for your component or if your component needs to perform special processing based on the number of open connections.

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

Related Sample Code
SoftVDigX

Declared In
`Components.h`

CountComponents

Returns the number of registered components that meet the selection criteria specified by your application.

```
long CountComponents (
   ComponentDescription *looking
);
```

Parameters

looking

A pointer to a `ComponentDescription` (page 50) structure. Your application specifies the criteria for the component search in the fields of this structure.

The Component Manager ignores fields in the component description structure that are set to 0. For example, if you set all the fields to 0, the Component Manager returns the number of components registered in the system. Similarly, if you set all fields to 0 except for the `componentManufacturer` field, the Component Manager returns the number of registered components supplied by the manufacturer you specify.

Return Value
The number of components that meet the specified search criteria.

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

Declared In
`Components.h`

CSGetComponentsThreadMode

Indicates whether using thread-unsafe components is allowed in the current thread.

```
CSComponentsThreadMode CSGetComponentsThreadMode (
   void
);
```

Return Value
A flag that indicates whether using thread-unsafe components is allowed in the current thread.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Components.h`

CSSetComponentsThreadMode

Sets whether or not using thread-unsafe components is allowed in the current thread.

```
void CSSetComponentsThreadMode (
   CSComponentsThreadMode mode
);
```

Parameters

mode

A flag that determines whether the current thread is restricted to calling components that are thread-safe. You should set this flag to `kCSAcceptThreadSafeComponentsOnlyMode` whenever you want the current thread to call only components that are thread-safe.

Discussion
Core Services maintains a component thread-mode flag for each thread in the current process. The default value of this flag is `kCSAcceptAllComponentsMode`, which means the thread can call any component regardless of whether the component is thread-safe. Applications and other high-level code that call component-based APIs (such as QuickTime) from preemptive threads should call this function from their thread beforehand and pass in the value `kCSAcceptThreadSafeComponentsOnlyMode`.

A thread's component thread-mode flag can safely retain its default value only if the thread is the main thread or if it participates in cooperative locking, such as Carbon Thread Manager-style cooperative threads and application threads that perform their own private locking.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Components.h`

DelegateComponentCall

Allows your component to pass on a request to a specified component.

```
ComponentResult DelegateComponentCall (
   ComponentParameters *originalParams,
   ComponentInstance ci
);
```

Parameters

originalParams

A pointer to the `ComponentParameters` (page 54) structure provided to your component by the Component Manager.

ci

The component instance that is to process the request. The Component Manager provides a component instance to your component when it opens a connection to another component with the `OpenComponent` (page 36) or `OpenDefaultComponent` (page 38) function. You must specify a component instance; this function does not accept a component identifier.

Return Value
The component result returned by the specified component.

Discussion
Your component may supplement its capabilities by using the services of another component to directly satisfy application requests using this function. For example, you might want to create two similar components that provide different levels of service to applications. Rather than completely implementing both components, you could design one to rely on the capabilities of the other. In this manner, you have to implement only that portion of the more capable component that provides additional services.

You may also invoke the services of another component using the standard mechanisms used by applications. The Component Manager then passes the requests to the appropriate component, and your component receives the results of those requests.

Your component must open a connection to the component to which the requests are to be passed. Your component must close that connection when it has finished using the services of the other component.

Your component should never use this function with open or close requests from the Component Manager—always use the `OpenComponent` and `CloseComponent` (page 19) functions to manage connections with other components.

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

Declared In
`Components.h`

DisposeComponentFunctionUPP

```
void DisposeComponentFunctionUPP (
   ComponentFunctionUPP userUPP
);
```

Parameters

userUPP

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

Declared In
`Components.h`

DisposeComponentMPWorkFunctionUPP

```
void DisposeComponentMPWorkFunctionUPP (
   ComponentMPWorkFunctionUPP userUPP
);
```

Parameters

userUPP

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

Declared In
`Components.h`

DisposeComponentRoutineUPP

Disposes of the universal procedure pointer (UPP) to a component routine callback function.

```
void DisposeComponentRoutineUPP (
   ComponentRoutineUPP userUPP
);
```

Parameters

userUPP

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

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

Declared In
`Components.h`

DisposeGetMissingComponentResourceUPP

```
void DisposeGetMissingComponentResourceUPP (
   GetMissingComponentResourceUPP userUPP
);
```

Parameters

userUPP

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

Declared In
`Components.h`

FindNextComponent

Returns the component identifier for the next registered component that meets the selection criteria specified by your application.

```
Component FindNextComponent (
   Component aComponent,
   ComponentDescription *looking
);
```

Parameters

aComponent

The starting point for the search. Set this field to 0 to start the search at the beginning of the component list. If you are continuing a search, you can specify a component identifier previously returned by this function. The function then searches the remaining components.

looking

A pointer to a `ComponentDescription` (page 50) structure. Your application specifies the criteria for the component search in the fields of this structure.

The Component Manager ignores fields in the component description structure that are set to 0. For example, if you set all the fields to 0, all components meet the search criteria. In this case, your application can retrieve information about all of the components that are registered in the system by repeatedly calling `FindNextComponent` and `GetComponentInfo` (page 26) until the search is complete. Similarly, if you set all fields to 0 except for the `componentManufacturer` field, the Component Manager searches all registered components for a component supplied by the manufacturer you specify. Note that this function does not modify the contents of the component description structure you supply. To retrieve detailed information about a component, you need to use the `GetComponentInfo` (page 26) function to get the component description structure for each returned component.

Return Value
The component identifier of a component that meets the search criteria or 0 when there are no more matching components. Your application can use the component identifier returned by this function to get more information about the component, using `GetComponentInfo`, or to open the component, using either the `OpenDefaultComponent` (page 38) function or the `OpenComponent` (page 36) function. See the description of the `Component` data type.

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

Related Sample Code
SoftVDigX
WhackedTV

Declared In
`Components.h`

GetComponentIconSuite

Returns a handle to a component's icon suite to your application.

```
OSErr GetComponentIconSuite (
   Component aComponent,
   Handle *iconSuite
);
```

Parameters

aComponent
The component whose icon suite you wish to obtain. Your application obtains a component identifier from the `FindNextComponent` (page 24) function. If your application registers a component, it can also obtain a component identifier from the `RegisterComponent` (page 38) or `RegisterComponentResource` (page 41) function. You can use a component instance here, but you must coerce the data type appropriately.

iconSuite
On return, a pointer to a handle for the component's icon suite or, if the component has not provided an icon suite, `NULL`. A component provides the resource ID of its icon family to the Component Manager in the optional extensions to the component resource. Your application is responsible for disposing of the returned icon suite handle.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Components.h`

GetComponentIndString

```
OSErr GetComponentIndString (
   Component aComponent,
   Str255 theString,
   SInt16 strListID,
   SInt16 index
);
```

Parameters

aComponent

theString

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SoftVDigX

Declared In

`Components.h`

GetComponentInfo

Returns to your application the registration information for a component.

```
OSErr GetComponentInfo (
   Component aComponent,
   ComponentDescription *cd,
   Handle componentName,
   Handle componentInfo,
   Handle componentIcon
);
```

Parameters

aComponent

The component about which you wish to obtain information. Your application obtains a component identifier from the `FindNextComponent` (page 24) function. If your application registers a component, it can also obtain a component identifier from the `RegisterComponent` (page 38) or `RegisterComponentResource` (page 41) function.

You may supply a component instance rather than a component identifier to this function, but you must coerce the data type appropriately. Your application can obtain a component instance from the `OpenComponent` (page 36) or `OpenDefaultComponent` (page 38) functions.

cd

A pointer to a `ComponentDescription` (page 50) structure. The function returns information about the specified component in this structure.

componentName

On return, a handle to the component's name. If the component does not have a name, an empty handle. Set this field to `NULL` if you do not want to receive the component's name.

componentInfo

On return, a handle to the component's information string. If the component does not have an information string, an empty handle. Set this field to `NULL` if you do not want to receive the component's information string.

componentIcon

On return, a handle to the component's icon. If the component does not have an icon, an empty handle. Set this field to `NULL` if you do not want to receive the component's icon. To get a handle to the component's icon suite, if it provides one, use the `GetComponentIconSuite` (page 25) function.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Discussion
For information on registering components, see "Registering Components".

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

Related Sample Code
WhackedTV

Declared In
`Components.h`

GetComponentInstanceError

Returns to your application the last error generated by a specific connection to a component.

```
OSErr GetComponentInstanceError (
   ComponentInstance aComponentInstance
);
```

Parameters

aComponentInstance

The component instance from which you want error information. Your application obtains the component instance from the `OpenDefaultComponent` (page 38) function or the `OpenComponent` (page 36) function. You can use a component identifier here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Discussion
Some component functions return error information as their function result. Other component functions set an error code that your application can retrieve using this function. Refer to the documentation supplied with the component for information on how that particular component handles errors.

Once you have retrieved an error code, the Component Manager clears the error code for the connection. If you want to retain that error value, you should save it in your application's local storage.

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

Declared In
`Components.h`

GetComponentInstanceStorage

Allows your component to retrieve a handle to the memory associated with a connection.

```
Handle GetComponentInstanceStorage (
   ComponentInstance aComponentInstance
);
```

Parameters

aComponentInstance

The connection for which to retrieve the associated memory. The Component Manager provides a component instance to your component when the connection is opened. You can use a component identifier here, but you must coerce the data type appropriately.

Return Value
A handle to the memory associated with the specified connection.

Discussion
Typically, your component does not need to use this function, because the Component Manager provides this handle to your component each time the client application requests service from this connection.

Your component tells the Component Manager about the memory associated with a connection by calling the `SetComponentInstanceStorage` (page 43) function.

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

Declared In
`Components.h`

GetComponentListModSeed

Allows your application to determine if the list of registered components has changed.

```
SInt32 GetComponentListModSeed (
   void
);
```

Parameters

Return Value
The component registration seed number. Each time the Component Manager registers or unregisters a component it generates a new, unique seed number. By comparing the return value to values previously returned by this function, you can determine whether the list has changed. Your application may use this information to rebuild its internal component lists or to trigger other activity that is necessary whenever new components are available.

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

Declared In
`Components.h`

GetComponentPublicIndString

```
OSErr GetComponentPublicIndString (
   Component aComponent,
   Str255 theString,
   SInt16 strListID,
   SInt16 index
);
```

Parameters

aComponent

theString

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Declared In

Components.h

GetComponentPublicResource

```
OSErr GetComponentPublicResource (
   Component aComponent,
   OSType resourceType,
   SInt16 resourceID,
   Handle *theResource
);
```

Parameters

aComponent

resourceType

theResource

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Declared In

Components.h

GetComponentPublicResourceList

```
OSErr GetComponentPublicResourceList (
   OSType resourceType,
   SInt16 resourceID,
   SInt32 flags,
   ComponentDescription *cd,
   GetMissingComponentResourceUPP missingProc,
   void *refCon,
   void *atomContainerPtr
);
```

Parameters

resourceType

cd

missingProc

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

GetComponentRefcon

Retrieves the value of the reference constant for your component.

```
long GetComponentRefcon (
   Component aComponent
);
```

Parameters

aComponent

The component whose reference constant you wish to get. You can use a component instance here, but you must coerce the data type appropriately.

Return Value

The reference constant for the specified component.

Discussion

There is one reference constant for each component, regardless of the number of connections to that component. When your component is registered, the Component Manager sets this reference constant to 0.

The reference constant is a 4-byte value that your component can use in any way you decide. For example, you might use the reference constant to store the address of a data structure that is shared by all connections maintained by your component. You should allocate shared structures in the system heap. Your component should deallocate the structure when its last connection is closed or when it is unregistered.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Components.h`

GetComponentResource

```
OSErr GetComponentResource (
   Component aComponent,
   OSType resType,
   SInt16 resID,
   Handle *theResource
);
```

Parameters

aComponent

resType

theResource

Return Value
A result code. See "Component Manager Result Codes" (page 69).

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

Declared In
`Components.h`

GetComponentTypeModSeed

```
SInt32 GetComponentTypeModSeed (
   OSType componentType
);
```

Parameters

componentType

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

Declared In
`Components.h`

InvokeComponentMPWorkFunctionUPP

```
ComponentResult InvokeComponentMPWorkFunctionUPP (
   void *globalRefCon,
   ComponentMPWorkFunctionHeaderRecordPtr header,
   ComponentMPWorkFunctionUPP userUPP
);
```

Parameters

header

userUPP

Return Value

See the description of the `ComponentResult` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

InvokeComponentRoutineUPP

Calls your component routine callback function

```
ComponentResult InvokeComponentRoutineUPP (
   ComponentParameters *cp,
   Handle componentStorage,
   ComponentRoutineUPP userUPP
);
```

Parameters

cp

componentStorage

userUPP

Return Value

See the description of the `ComponentResult` data type.

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

InvokeGetMissingComponentResourceUPP

```
OSErr InvokeGetMissingComponentResourceUPP (
   Component c,
   OSType resType,
   SInt16 resID,
   void *refCon,
   Handle *resource,
   GetMissingComponentResourceUPP userUPP
);
```

Parameters

c

resType

resource

userUPP

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

NewComponentFunctionUPP

```
ComponentFunctionUPP NewComponentFunctionUPP (
   ProcPtr userRoutine,
   ProcInfoType procInfo
);
```

Parameters

procInfo

Return Value

See the description of the `ComponentFunctionUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

NewComponentMPWorkFunctionUPP

```
ComponentMPWorkFunctionUPP NewComponentMPWorkFunctionUPP (
   ComponentMPWorkFunctionProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ComponentMPWorkFunctionUPP` data type.

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

Declared In
`Components.h`

NewComponentRoutineUPP

Creates a new universal procedure pointer (UPP) to a component routine callback function.

```
ComponentRoutineUPP NewComponentRoutineUPP (
   ComponentRoutineProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ComponentRoutineUPP` data type.

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

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

Declared In
`Components.h`

NewGetMissingComponentResourceUPP

```
GetMissingComponentResourceUPP NewGetMissingComponentResourceUPP (
   GetMissingComponentResourceProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `GetMissingComponentResourceUPP` data type.

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

Declared In
`Components.h`

OpenAComponent

```
OSErr OpenAComponent (
   Component aComponent,
   ComponentInstance *ci
);
```

Parameters

aComponent

ci

Return Value
A result code. See "Component Manager Result Codes" (page 69).

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

Declared In
`Components.h`

OpenAComponentResFile

```
OSErr OpenAComponentResFile (
   Component aComponent,
   ResFileRefNum *resRef
);
```

Parameters

aComponent

Return Value
A result code. See "Component Manager Result Codes" (page 69).

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

Declared In
`Components.h`

OpenADefaultComponent

```
OSErr OpenADefaultComponent (
   OSType componentType,
   OSType componentSubType,
   ComponentInstance *ci
);
```

Parameters

componentType

componentSubType

ci

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

WhackedTV

Declared In

`Components.h`

OpenComponent

Opens a connection to the component with the component identifier specified by your application.

```
ComponentInstance OpenComponent (
   Component aComponent
);
```

Parameters

aComponent

The component you wish to open. Your application obtains this identifier from the `FindNextComponent` (page 24) function. If your application registers a component, it can also obtain a component identifier from the `RegisterComponent` function or the `RegisterComponentResource` function.

Return Value

A component instance which identifies your application's connection to the component. You must supply this component instance whenever you call the functions provided by the component. When you close the component, you must also supply this component instance to the `CloseComponent` (page 19) function.

If it cannot open the specified component, the function returns `NULL`.

See the description of the `ComponentInstance` data type.

Discussion

Your application must open a component before it can call any component functions. To use this function, you must already have obtained a component identifier. Alternatively, you can use the `OpenDefaultComponent` (page 38) function to open a component without calling `FindNextComponent`.

Note that your application may maintain several connections to a single component, or it may have connections to several components at the same time.

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

Related Sample Code
WhackedTV

Declared In
`Components.h`

OpenComponentResFile

Allows your component to gain access to its resource file.

```
ResFileRefNum OpenComponentResFile (
   Component aComponent
);
```

Parameters

aComponent

The component whose resource file you wish to open. Applications that register components may obtain this identifier from the `RegisterComponentResource` (page 41) function. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
A reference number that your component can use to read data from the appropriate resource file. If the specified component does not have an associated resource file or if the Component Manager cannot open the resource file, the function returns 0 or a negative number.

Discussion
This function opens the resource file with read-only permission. The Component Manager adds the resource file to the current resource chain. Your component must close the resource file with the `CloseComponentResFile` (page 19) function before returning to the calling application. Note that there is only one resource file associated with a component.

Your component can use `FSpOpenResFile` or equivalent Resource Manager functions to open other resource files, but you must use this function to open your component's resource file.

If you store your component in a component resource but register the component with the `RegisterComponent` (page 38) function, rather than with the `RegisterComponentResource` or `RegisterComponentResourceFile` function, your component cannot access its resource file with this function.

Note that when working with resources, your component should always first save the current resource file, perform any resource operations, then restore the current resource file to its previous value before returning.

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

Declared In
`Components.h`

OpenDefaultComponent

Opens a connection to a registered component of the component type and subtype specified by your application.

```
ComponentInstance OpenDefaultComponent (
   OSType componentType,
   OSType componentSubType
);
```

Parameters

componentType

The type of the component. All components of a particular type support a common set of interface functions. Use this parameter to search for components of a given type.

componentSubType

The subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard functions for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor.

Your application can use the `componentSubType` parameter to perform a more specific lookup operation than is possible using only the `componentType` parameter. For example, you may want your application to use only components of a certain component type (`'draw'`) that also have a specific subtype (`'oval'`). Set this parameter to 0 to select a component with any subtype value.

Return Value

A component instance that identifies the connection opened to the component which matches your search criteria. You must supply this component instance whenever you call the functions provided by the component. When you close the component, you must also supply this component instance to the `CloseComponent` (page 19) function.

If more than one component in the list of registered components meets the search criteria, the function opens the first one that it finds in its list. If it cannot open the specified component, it returns `NULL`.

See the description of the `ComponentInstance` data type.

Discussion

Your application must open a component before it can call any component functions. This function searches for a component by type and subtype. You do not have to supply a component description structure or call the `FindNextComponent` (page 24) function to use this function. If you want to exert more control over the selection process, you can use the `FindNextComponent` and `OpenComponent` (page 36) functions.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

RegisterComponent

Registers a component stored in memory.

```
Component RegisterComponent (
   ComponentDescription *cd,
   ComponentRoutineUPP componentEntryPoint,
   SInt16 global,
   Handle componentName,
   Handle componentInfo,
   Handle componentIcon
);
```

Parameters

cd

A pointer to a `ComponentDescription` (page 50) structure that describes the component to be registered. You must correctly fill in the fields of this structure before calling this function. When applications search for components using the `FindNextComponent` (page 24) function, the Component Manager compares the attributes you specify here with those specified by the application. If the attributes match, the Component Manager returns the component identifier to the application.

componentEntryPoint

A universal procedure pointer (UPP) to your component's main entry point. The function referred to by this parameter receives all requests for the component. See the `ComponentRoutineProcPtr` (page 47) callback for more information on creating a component function.

global

A set of flags that control the scope of component registration. See Register Component Resource flags (page 66) for a description of the flags.

componentName

A handle to the component's name. Set this parameter to `NULL` if you do not want to assign a name to the component.

componentInfo

A handle to the component's information string. Set this parameter to `NULL` if you do not want to assign an information string to the component.

componentIcon

A handle to the component's icon (a 32-by-32 pixel black-and-white icon). Set this parameter to `NULL` if you do not want to supply an icon for this component. Note that this icon is not used by the Finder you supply an icon only so that other components or applications can display your component's icon if needed.

Return Value

The unique component identifier assigned to the component by the Component Manager or, if it cannot register the component, `NULL`. See the description of the `Component` data type.

Discussion

Before a component can be used by an application, the component must be registered with the Component Manager. Applications can then find and open the component using the standard Component Manager functions.

Components you register with the `RegisterComponent` function must be in memory when you call this function. If you want to register a component that is stored in the resource fork of a file, use the `RegisterComponentResource` (page 41) function. Use the `RegisterComponentResourceFile` (page 42) function to register all components in the resource fork of a file. The Component Manager automatically registers component resources stored in files with file types of `'thng'` that are stored in the Extensions folder. See "Resources" for more information on component resource files.

Note that a component residing in your application heap remains registered until your application unregisters it or quits. When an application quits, the Component Manager automatically closes any component connections to that application. In addition, if the application has registered components that reside in its heap space, the Component Manager automatically unregisters those components. A component residing in the system heap and registered by your application remains registered until your application unregisters it or until the computer is shut down.

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

Declared In
`Components.h`

RegisterComponentFileRef

```
OSErr RegisterComponentFileRef (
   const FSRef *ref,
   SInt16 global
);
```

Parameters

ref

Return Value
A result code. See "Component Manager Result Codes" (page 69).

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

Declared In
`Components.h`

RegisterComponentFileRefEntries

```
OSErr RegisterComponentFileRefEntries (
   const FSRef *ref,
   SInt16 global,
   const ComponentDescription *toRegister,
   UInt32 registerCount
);
```

Parameters

ref

toRegister

registerCount

Return Value
A result code. See "Component Manager Result Codes" (page 69).

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

Declared In
`Components.h`

RegisterComponentResource

Registers a component stored in a resource file.

```
Component RegisterComponentResource (
   ComponentResourceHandle cr,
   SInt16 global
);
```

Parameters

cr

A handle to a component resource that describes the component to be registered. The component resource contains all the information required to register the component. Components you register with this function must be stored in a resource file as a component resource. The Component Manager automatically registers component resources stored in files with file types of `'thng'` that are stored in the Extensions folder. See "Resources" for more information on component resource files.

global

A set of flags that controls the scope of component registration. See Register Component Resource flags (page 66) for a description of the flags.

Return Value
The unique component identifier assigned to the component by the Component Manager, or `NULL` if the function could not register the component. See the description of the `Component` data type.

Discussion
Before a component can be used by an application, the component must be registered with the Component Manager. Applications can then find and open the component using the standard Component Manager functions.

If you want to register a component that is in memory, use the `RegisterComponent` (page 38) function.

This function does not actually load the code specified by the component resource into memory. Rather, the Component Manager loads the component code the first time an application opens the component. If the code is not in the same file as the component resource or if the Component Manager cannot find the file, the open request fails.

Note that a component registered locally by your application remains registered until your application unregisters it or quits. When an application quits, the Component Manager automatically closes any component connections to that application. In addition, if the application has registered components that reside in its heap space, the Component Manager automatically unregisters those components. A component registered globally by your application remains registered until your application unregisters it or until the computer is shut down.

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

Declared In
`Components.h`

RegisterComponentResourceFile

Registers all component resources in the given resource file.

```
SInt32 RegisterComponentResourceFile (
   SInt16 resRefNum,
   SInt16 global
);
```

Parameters

resRefNum

The reference number of the resource file containing the components to register.

global

A set of flags that control the scope of the registration of the components in the resource file. See Register Component Resource flags (page 66) for a description of the flags.

Return Value

The number of components registered, if all components in the specified resource file are successfully registered. If one or more of the components in the resource file could not be registered, or if the specified file reference number is invalid, a negative function result.

Discussion

Before a component can be used by an application, the component must be registered with the Component Manager. The Component Manager automatically registers component resources stored in files with file types of `'thng'` that are stored in the Extensions folder. For a description of the format and content of component resources, see "Resources".

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ResolveComponentAlias

```
Component ResolveComponentAlias (
   Component aComponent
);
```

Parameters

aComponent

Return Value

See the description of the `Component` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

SetComponentInstanceError

Passes error information to the Component Manager which sets the current error value for the appropriate connection.

```
void SetComponentInstanceError (
   ComponentInstance aComponentInstance,
   OSErr theError
);
```

Parameters

aComponentInstance

The connection for which to set the error. The Component Manager provides a component instance to your component when the connection is opened. The Component Manager also provides a component instance to your component as the first parameter in the `params` field of the parameters structure.

theError

The new value for the current error.

Discussion

In general, your component returns error information in its function result a nonzero function result indicates an error occurred, and a function result of 0 indicates the request was successful. However, some requests require that your component return other information as its function result. In these cases, your component can use this function to report its latest error state to the Component Manager. You can also use this function at any time during your component's execution to report an error.

Applications retrieve this error information by calling the `GetComponentInstanceError` (page 27) function. The documentation for your component should specify how the component indicates errors.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

SetComponentInstanceStorage

Allows your component to associate memory with a connection.

```
void SetComponentInstanceStorage (
   ComponentInstance aComponentInstance,
   Handle theStorage
);
```

Parameters

aComponentInstance

The connection to associate with the allocated memory. The Component Manager provides a component instance to your component when the connection is opened. You can use a component identifier here, but you must coerce the data type appropriately.

theStorage

A handle to the memory that your component has allocated for the connection. Your component must allocate this memory in the current heap. The Component Manager saves this handle and provides it to your component, along with other parameters, in subsequent requests to this connection.

Discussion
When an application or component opens a connection to your component, the Component Manager sends your component an open request. In response to this open request, your component should set up an environment to service the connection. Typically, your component should allocate some memory for the connection. Your component can then use that memory to maintain state information appropriate to the connection.

Your component should dispose of any allocated memory for the connection only in response to the close request. Note that whenever an open request fails, the Component Manager always issues the close request. Furthermore, the value stored with this function is always passed to the close request, so it must be valid or `NULL`. If the open request tries to dispose of its allocated memory before returning, it should call this function again with a `NULL` handle to keep the Component Manager from passing an invalid handle to the close request.

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

Related Sample Code
SoftVDigX

Declared In
`Components.h`

SetComponentRefcon

Sets the reference constant for your component.

```
void SetComponentRefcon (
   Component aComponent,
   long theRefcon
);
```

Parameters

aComponent
The component whose reference constant you wish to set. You can use a component instance here, but you must coerce the data type appropriately.

theRefcon
The reference constant value that you want to set for your component. Your component can retrieve the reference constant using the `GetComponentRefcon` (page 30) function.

Discussion
There is one reference constant for each component, regardless of the number of connections to that component. When your component is registered, the Component Manager sets this reference constant to 0.

The reference constant is a 4-byte value that your component can use in any way you decide. For example, you might use the reference constant to store the address of a data structure that is shared by all connections maintained by your component. You should allocate shared structures in the system heap. Your component should deallocate the structure when its last connection is closed or when it is unregistered.

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

Declared In
`Components.h`

SetDefaultComponent

Changes the search order for registered components.

```
OSErr SetDefaultComponent (
   Component aComponent,
   SInt16 flags
);
```

Parameters

aComponent

The component which you wish moved to the front of the search chain. The order of the search chain influences which component the Component Manager selects in response to an application's use of the `OpenDefaultComponent` (page 38) and `FindNextComponent` (page 24) functions. You can use a component instance here, but you must coerce the data type appropriately.

flags

A value specifying the control information governing the operation. The value of this parameter controls which component description fields the Component Manager examines during the reorder operation. Set the appropriate flags to 1 to define the fields that are examined during the reorder operation. See Set Default Component Flags (page 68) for a description of the values you can use here.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Discussion
Note that this function changes the search order for all applications. As a result, you should use this function carefully.

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

Declared In
`Components.h`

UncaptureComponent

Allows your component to uncapture a previously captured component.

```
OSErr UncaptureComponent (
   Component aComponent
);
```

Parameters

aComponent

The component to be uncaptured. Your component obtains this identifier from the `CaptureComponent` (page 18) function. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Discussion
This function restores the specified component to the list of available components. Applications can then access the component and retrieve information about the component using Component Manager functions.

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

Declared In
`Components.h`

UnregisterComponent

Removes a component from the Component Manager's registration list.

```
OSErr UnregisterComponent (
   Component aComponent
);
```

Parameters

aComponent

The component to be removed. Applications that register components may obtain this identifier from the `RegisterComponent` (page 38) or `RegisterComponentResource` (page 41) functions. The component must not be in use by any applications or components. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 69). If there are open connections to the component, returns a `validInstancesExist` error.

Discussion
Most components are registered at startup and remain registered until the computer is shut down. However, you may want to provide some services temporarily. In that case you dispose of the component that provides the temporary service by using this function.

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

Declared In
`Components.h`

Callbacks

ComponentMPWorkFunctionProcPtr

```
typedef ComponentResult (*ComponentMPWorkFunctionProcPtr) (
   void * globalRefCon,
   ComponentMPWorkFunctionHeaderRecordPtr header
);
```

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

```
ComponentResult MyComponentMPWorkFunctionProc (
```

```
   void * globalRefCon,
   ComponentMPWorkFunctionHeaderRecordPtr header
);
```

Parameters

header

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

ComponentRoutineProcPtr

Defines a pointer to your component callback function, which serves as the main entry point into your component and performs the component's services.

```
typedef ComponentResult (*ComponentRoutineProcPtr) (
   ComponentParameters * cp,
   Handle componentStorage
);
```

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

```
ComponentResult ComponentRoutineProcPtr (
   ComponentParameters * cp,
   Handle componentStorage
);
```

Parameters

cp

A `ComponentParameters` (page 54) structure. The `what` field of the component parameters structure indicates the action your component should perform. The parameters that the client invoked your function with are contained in the `params` field of the component parameters structure. Your component can use the `CallComponentFunction` (page 13) or `CallComponentFunctionWithStorage` (page 14) function to extract the parameters from this structure.

componentStorage

A handle to any memory that your component has associated with the connection. Typically, upon receiving an open request, your component allocates memory and uses the `SetComponentInstanceStorage` (page 43) function to associate the allocated memory with the component connection.

Return Value
Your component should return a value of type `ComponentResult`. If your component does not return error information as its function result, it should indicate errors using the `SetComponentInstanceError` (page 43) function. See the description of the `ComponentResult` data type.

Discussion
You pass a pointer to your component callback function to the Component Manager when you register your component. The Component Manager can then call your component when another application or component requests its services. When your component receives a request, it should perform the action specified in the `what` field of the component parameters structure.

The pointer which you pass to the Component Manager should be a universal procedure pointer (UPP). The definition of the UPP data type for your component function is as follows:

```
typedef (ComponentRoutineProcPtr) ComponentRoutineUPP;
```

Before using your component function, you must first create a UPP for your callback function, using the `NewComponentRoutineUPP` (page 34) function, as shown here:

```
ComponentRoutineUPP MyComponentRoutineUPP;
MyComponentRoutineUPP =
NewComponentRoutineUPP(&MyComponentRoutineProc)
```

You then pass `MyComponentRoutineUPP` to the Component Manager when you register your component. The Component Manager will call your function each time your component receives a request. If you wish to call your component function yourself, you can use the `InvokeComponentRoutineUPP` (page 32) function.

```
result = InvokeComponentRoutineUPP &myParams, myStorage,
MyComponentRoutineUPP)
```

When you are finished with your component callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeComponentRoutineUPP` (page 23) function.

```
DisposeComponentRoutineUPP(MyComponentRoutineUPP);
```

To provide a component, you define a component function and supply the appropriate registration information. You store your component function in a code resource and typically store your component's registration information as resources in a component file.

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

Declared In
`Components.h`

GetMissingComponentResourceProcPtr

```
typedef OSErr (*GetMissingComponentResourceProcPtr) (
   Component c,
   OSType resType,
   short resID,
   void * refCon,
   Handle * resource
);
```

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

```
OSErr GetMissingComponentResourceProcPtr (
   Component c,
   OSType resType,
```

```
    short resID,
    void * refCon,
    Handle * resource
);
```

Parameters

c

resType

resource

Return Value

A result code. See "Component Manager Result Codes" (page 69).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

Data Types

ComponentAliasResource

```
struct ComponentAliasResource {
    ComponentResource cr;
    ComponentDescription aliasCD;
};
typedef struct ComponentAliasResource ComponentAliasResource;
```

Fields

`cr`

`aliasCD`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ComponentDependencyArray

```
struct ComponentDependencyArray {
    SInt32 count;
    ComponentDescription descArray[1];
};
```

Fields

`count`

`descArray`

ComponentDescription

```
struct ComponentDescription {
    OSType componentType;
    OSType componentSubType;
    OSType componentManufacturer;
    unsigned long componentFlags;
    unsigned long componentFlagsMask;
};
typedef struct ComponentDescription ComponentDescription;
```

Fields

`componentType`

A four-character code that identifies the type of component. All components of a particular type must support a common set of interface functions. For example, drawing components all have a component type of `'draw'`.

If you are developing an application which uses components, you can use this field to search for components of a given type by specifying the component type in this field of the component description structure you supply to the `FindNextComponent` (page 24) function or the `CountComponents` (page 20) function. A value of 0 operates as a wildcard.

If you are developing a component, it must support all of the standard functions for the component type specified by this field. Type codes with all lowercase characters are reserved for definition by Apple. You can define your own component type code as long as you register it with Apple's Component Registry Group.

`componentSubType`

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard functions for a given component type. For example, the subtype of drawing components indicates the type of object the component draws. Drawing components that draw ovals have a subtype of `'oval'`.

If you are developing an application which uses components, you can use the `componentSubType` field to perform a more specific lookup operation than is possible using only the `componentType` field. By specifying particular values for both fields in the component description structure that you supply to the `FindNextComponent` or `CountComponents` function, your application retrieves information about only those components that meet both of these search criteria. A value of 0 operates as a wildcard.

If you are developing a component, you may use this field to indicate more specific information about the capabilities of the component. There are no restrictions on the content you assign to this field. If no additional information is appropriate for your component type, you may set the `componentSubType` field to 0.

`componentManufacturer`

A four-character code that identifies the manufacturer of the component. This field allows for further differentiation between individual components. For example, components made by a specific manufacturer may support an extended feature set. Components provided by Apple use a manufacturer value of `'appl'`.

If you are developing an application which uses components, you can use this field to find components from a certain manufacturer. Specify the appropriate manufacturer code in this field of the component description structure you supply to the `FindNextComponent` or `CountComponents` function. A value of 0 operates as a wildcard.

If you are developing a component, you obtain your manufacturer code, which can be the same as your application signature, from Apple's Component Registry Group.

`componentFlags`

A 32-bit field that provides additional information about a particular component.

The high-order 8 bits are reserved for definition by the Component Manager. If you are developing an application, you should usually set these bits to 0.

The low-order 24 bits are specific to each component type. These flags can be used to indicate the presence of features or capabilities in a given component.

If you are developing an application which uses components, you can use these flags to further narrow the search criteria applied by the `FindNextComponent` or `CountComponents` function. If you use the `componentFlags` field in a component search, you use the `componentFlagsMask` field to indicate which flags are to be considered in the search.

If you are developing a component, you can use these flags to indicate any special capabilities or features of your component. You may use all 24 bits, as appropriate to its component type. You must set all unused bits to 0.

`componentFlagsMask`

A 32-bit field that indicates which flags in the `componentFlags` field are relevant to a particular component search operation.

If you are developing an application which uses components, your application should set each bit which corresponds to a flag in the `componentFlags` field that is to be considered as a search criterion by the `FindNextComponent` or `CountComponents` function to 1. The Component Manager considers only these flags during the search. You specify the desired flag value (either 0 or 1) in the `componentFlags` field.

For example, to look for a component with a specific control flag that is set to 0, set the appropriate bit in the `ComponentFlags` field to 0 and the same bit in the `ComponentFlagsMask` field to 1. To look for a component with a specific control flag that is set to 1, set the bit in the `ComponentFlags` field to 1 and the same bit in the `ComponentFlagsMask` field to 1. To ignore a flag, set the bit in the `ComponentFlagsMask` field to 0.

If you are developing a component, your component must set the `componentFlagsMask` field in its component description structure to 0.

Discussion

The ComponentDescription structure identifies the characteristics of a component, including the type of services offered by the component and its manufacturer.

Applications and components use component description structures in different ways. An application that uses components specifies the selection criteria for a component in a component description structure. The functions `FindNextComponent` (page 24) , `CountComponents` (page 20) , and `GetComponentInfo` (page 26) all use the component description structure to specify the criteria for their search.

A component uses the component description structure to specify its registration information and capabilities and identify itself to the Component Manager. If your component is stored in a component resource, the information in the component description structure must be part of that resource. See the description of the component 'thng' resource. If you have developed an application that registers your component, that application must supply a component description structure to the `RegisterComponent` (page 38) function. See "Registering Components" for information about registering components.

The `ComponentDescription` data type defines the component description structure. Note that the valid values of fields in the component description structure are determined by the component type specification. For example, all image compressor components must use the `componentSubType` field to specify the compression algorithm used by the compressor.

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

Declared In
`Components.h`

ComponentFunctionUPP

```
typedef UniversalProcPtr ComponentFunctionUPP;
```

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

Declared In
`Components.h`

ComponentInstanceRecord

```
struct ComponentInstanceRecord {
    long data[1];
};
typedef struct ComponentInstanceRecord ComponentInstanceRecord;
typedef ComponentInstanceRecord * ComponentInstance;
```

Fields
`data`

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

Declared In
`Components.h`

ComponentMPWorkFunctionHeaderRecord

```
struct ComponentMPWorkFunctionHeaderRecord {
    UInt32 headerSize;
    UInt32 recordSize;
    UInt32 workFlags;
    UInt16 processorCount;
    UInt8 unused;
    UInt8 isRunning;
};
typedef struct ComponentMPWorkFunctionHeaderRecord
ComponentMPWorkFunctionHeaderRecord;
typedef ComponentMPWorkFunctionHeaderRecord *
ComponentMPWorkFunctionHeaderRecordPtr;
```

Fields

`headerSize`

`recordSize`

`workFlags`

`processorCount`

`unused`

`isRunning`

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

Declared In
`Components.h`

ComponentMPWorkFunctionUPP

```
typedef ComponentMPWorkFunctionProcPtr ComponentMPWorkFunctionUPP;
```

Discussion
For more information, see the description of the ComponentMPWorkFunctionUPP callback function.

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

Declared In
`Components.h`

ComponentParameters

```
struct ComponentParameters {
    UInt8 flags;
    UInt8 paramSize;
    short what;
    long params[1];
};
typedef struct ComponentParameters ComponentParameters;
```

Fields

`flags`

Reserved for use by Apple.

`paramSize`

Specifies the number of bytes of parameter data for this request. The actual parameters are stored in the `params` field.

`what`

Specifies the type of request. Component designers define the meaning of positive values and assign them to requests that are supported by components of a given type. Negative values are reserved for definition by Apple. See "Result Codes" for Apple-defined request code values.

`params`

An array that contains the parameters specified by the application that called your component. You can use the `CallComponentRoutine` or `CallComponentRoutineWithStorage` function to convert this array into a Pascal-style invocation of a subroutine in your component.

Discussion

The Component Manager uses the component parameters structure to pass information to your component about a request from an application. Functions which use this data type are `CallComponentFunction` (page 13) , `CallComponentFunctionWithStorage` (page 14) , and `DelegateComponentCall` (page 22).The information in this structure completely defines the request. Your component services the request as appropriate.

The `ComponentParameters` data type defines the component parameters structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ComponentPlatformInfo

```
struct ComponentPlatformInfo {
    long componentFlags;
    ResourceSpec component;
    short platformType;
};
typedef struct ComponentPlatformInfo ComponentPlatformInfo;
```

Fields

`component`

Availability

Available in Mac OS X v10.0 and later.

Declared In
Components.h

ComponentPlatformInfoArray

```
struct ComponentPlatformInfoArray {
    long count;
    ComponentPlatformInfo platformArray[1];
};
typedef struct ComponentPlatformInfoArray ComponentPlatformInfoArray;
```

Fields
platformArray

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

Declared In
Components.h

ComponentRecord

```
struct ComponentRecord {
    long data[1];
};
typedef struct ComponentRecord ComponentRecord;
typedef ComponentRecord * Component;
```

Fields
data

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

Declared In
Components.h

ComponentResource

```
struct ComponentResource {
    ComponentDescription cd;
    ResourceSpec component;
    ResourceSpec componentName;
    ResourceSpec componentInfo;
    ResourceSpec componentIcon;
};
typedef struct ComponentResource ComponentResource;
typedef ComponentResource * ComponentResourcePtr;
```

Fields
cd

A ComponentDescription (page 50) structure that specifies the characteristics of the component.

`component`

A resource specification structure that specifies the type and ID of the component code resource. The `resType` field of the resource specification structure may contain any value. The component's main entry point must be at offset 0 in the resource.

`componentName`

A resource specification structure that specifies the resource type and ID for the name of the component. This is a Pascal string. Typically, the name is stored in a resource of type `'STR '`.

`componentInfo`

A resource specification structure that specifies the resource type and ID for the information string that describes the component. This is a Pascal string. Typically, the information string is stored in a resource of type `'STR '`. You might use the information stored in this resource in a Get Info dialog box.

`componentIcon`

A resource specification structure that specifies the resource type and ID for the icon for a component. Component icons are stored as 32-by-32 bit maps. Typically, the icon is stored in a resource of type `'ICON'`. Note that this icon is not used by the Finder you supply an icon only so that other components or applications can display your component's icon in a dialog box if needed.

Discussion

The `ComponentResource` data type defines the structure of a component resource. You can also optionally append to the end of this structure the information defined by the `ComponentResourceExtension` (page 56) data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ComponentResourceExtension

```
struct ComponentResourceExtension {
    long componentVersion;
    long componentRegisterFlags;
    short componentIconFamily;
};
typedef struct ComponentResourceExtension ComponentResourceExtension;
```

Fields

`componentVersion`

The version number of the component. If you specify the `componentDoAutoVersion` flag in `componentRegisterFlags`, the Component Manager must obtain the version number of your component when your component is registered. Either you can provide a version number in your component's resource, or you can specify a value of 0 for its version number. If you specify 0, the Component Manager sends your component a version request to get the version number of your component.

`componentRegisterFlags`

A set of flags containing additional registration information. See Component Resource Extension Flags (page 61) for the flag values.

`componentIconFamily`

The resource ID of an icon family. You can provide an icon family in addition to the icon provided in the `componentIcon` field. Note that members of this icon family are not used by the Finder you supply an icon family only so that other components or applications can display your component's icon in a dialog box if needed.

Discussion
You can optionally include in your component resource the information defined by the `ComponentResourceExtension` data type:

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

Declared In
`Components.h`

ComponentResult

```
typedef long ComponentResult;
```

Discussion

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

Declared In
`Components.h`

ComponentRoutineUPP

```
typedef ComponentRoutineProcPtr ComponentRoutineUPP;
```

Discussion
For more information, see the description of the ComponentRoutineUPP callback function.

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

Declared In
`Components.h`

ExtComponentResource

```
struct ExtComponentResource {
    ComponentDescription cd;
    ResourceSpec component;
    ResourceSpec componentName;
    ResourceSpec componentInfo;
    ResourceSpec componentIcon;
    long componentVersion;
    long componentRegisterFlags;
    short componentIconFamily;
    long count;
    ComponentPlatformInfo platformArray[1];
};
typedef struct ExtComponentResource ExtComponentResource;
typedef ExtComponentResource * ExtComponentResourcePtr;
```

Fields

`cd`

`component`

`componentName`

`componentInfo`

`componentIcon`

`platformArray`

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

Declared In
`Components.h`

GetMissingComponentResourceUPP

```
typedef GetMissingComponentResourceProcPtr GetMissingComponentResourceUPP;
```

Discussion
For more information, see the description of the GetMissingComponentResourceUPP callback function.

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

Declared In
`Components.h`

RegisteredComponentInstanceRecord

```
struct RegisteredComponentInstanceRecord {
    long data[1];
};
typedef struct RegisteredComponentInstanceRecord  RegisteredComponentInstanceRecord;
typedef RegisteredComponentInstanceRecord *  RegisteredComponentInstanceRecordPtr;
```

Fields

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

RegisteredComponentRecord

```
struct RegisteredComponentRecord {
    long data[1];
};
typedef struct RegisteredComponentRecord RegisteredComponentRecord;
typedef RegisteredComponentRecord * RegisteredComponentRecordPtr;
```

Fields

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ResourceSpec

```
struct ResourceSpec {
    OSType resType;
    short resID;
};
typedef struct ResourceSpec ResourceSpec;
```

Fields

`resType`

The type of the resource.

`resID`

The ID of the resource.

Discussion

The `ComponentResource` (page 55) structure uses the resource specification structure, defined by the `ResourceSpec` data type, to describe the component's code, name, information string, or icon. The resources specified by the resource specification structures must reside in the same resource file as the component resource itself.

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

Declared In
`Components.h`

Constants

cmpAliasNoFlags

```
enum {
    cmpAliasNoFlags = 0,
    cmpAliasOnlyThisFile = 1
};
```

Constants

`cmpAliasNoFlags`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`cmpAliasOnlyThisFile`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

cmpIsMissing

```
enum {
    cmpThreadSafe = 1L << 28,
    cmpIsMissing = 1L << 29,
    cmpWantsRegisterMessage = 1L << 31
};
```

Constants

`cmpThreadSafe`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

`cmpIsMissing`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`cmpWantsRegisterMessage`
The setting of the `cmpWantsRegisterMessage` bit determines whether the Component Manager calls this component during registration. Set this bit to 1 if your component should be called when it is registered; otherwise, set this bit to 0.
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

Discussion
These values are used by the `componentFlags` field of the `ComponentDescription` (page 50) structure to provide additional information about a component.

Component Resource Extension Flags

```
enum {
    componentDoAutoVersion = (1 << 0),
    componentWantsUnregister = (1 << 1),
    componentAutoVersionIncludeFlags = (1 << 2),
    componentHasMultiplePlatforms = (1 << 3),
    componentLoadResident = (1 << 4)
};
```

Constants

`componentDoAutoVersion`
Specify this flag if you want the Component Manager to resolve conflicts between different versions of the same component. If you specify this flag, the Component Manager registers your component only if there is no later version available. If an older version is already registered, the Component Manager unregisters it. If a newer version of the same component is registered after yours, the Component Manager automatically unregisters your component. You can use this automatic version control feature to make sure that the most recent version of your component is registered, regardless of the number of versions that are installed

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentWantsUnregister`
Specify this flag if you want your component to receive an unregister request when it is unregistered.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentAutoVersionIncludeFlags`
Specify this flag if you want the Component Manager to include the `componentFlags` field of the component description structure when it searches for identical components in the process of performing automatic version control for your component. If you do not specify this flag, the Component Manager searches only the `componentType`, `componentSubType`, and `componentManufacturer` fields.

Note that the setting of the `componentAutoVersionIncludeFlags` flag affects automatic version control only and does not affect the search operations performed by `FindNextComponent` and `CountComponents`.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentHasMultiplePlatforms`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentLoadResident`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion
These values are used in the `ComponentResourceExtension` (page 56) structure to specify additional information about component registration.

CSComponentsThreadMode

```
typedef UInt32 CSComponentsThreadMode;
enum {
    kCSAcceptAllComponentsMode = 0,
    kCSAcceptThreadSafeComponentsOnlyMode = 1
};
```

Constants

`kCSAcceptAllComponentsMode`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

`kCSAcceptThreadSafeComponentsOnlyMode`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

kAnyComponentType

```
enum {
    kAnyComponentType = 0,
    kAnyComponentSubType = 0,
    kAnyComponentManufacturer = 0,
    kAnyComponentFlagsMask = 0
};
```

Constants

`kAnyComponentType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kAnyComponentSubType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kAnyComponentManufacturer`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kAnyComponentFlagsMask`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

kAppleManufacturer

```
enum {
    kAppleManufacturer = 'appl',
    kComponentResourceType = 'thng',
    kComponentAliasResourceType = 'thga'
};
```

Constants

`kAppleManufacturer`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kComponentResourceType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kComponentAliasResourceType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

mpWorkFlagDoWork

```
enum {
    mpWorkFlagDoWork = (1 << 0),
    mpWorkFlagDoCompletion = (1 << 1),
    mpWorkFlagCopyWorkBlock = (1 << 2),
    mpWorkFlagDontBlock = (1 << 3),
    mpWorkFlagGetProcessorCount = (1 << 4),
    mpWorkFlagGetIsRunning = (1 << 6)
};
```

Constants

`mpWorkFlagDoWork`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagDoCompletion`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagCopyWorkBlock`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagDontBlock`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagGetProcessorCount`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagGetIsRunning`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

platform68k

```
enum {
    platform68k = 1,
    platformPowerPC = 2,
    platformInterpreted = 3,
    platformWin32 = 4,
    platformPowerPCNativeEntryPoint = 5,
    platformIA32NativeEntryPoint = 6
};
```

Constants

`platform68k`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformPowerPC`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformInterpreted`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformWin32`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformPowerPCNativeEntryPoint`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformIA32NativeEntryPoint`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

Discussion

platformIRIXmips

```
enum {
    platformIRIXmips = 1000,
    platformSunOSsparc = 1100,
    platformSunOSintel = 1101,
    platformLinuxppc = 1200,
    platformLinuxintel = 1201,
    platformAIXppc = 1300,
    platformNeXTIntel = 1400,
    platformNeXTppc = 1401,
    platformNeXTsparc = 1402,
    platformNeXT68k = 1403,
    platformMacOSx86 = 1500
};
```

Constants

`platformIRIXmips`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformSunOSsparc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformSunOSintel`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformLinuxppc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformLinuxintel`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformAIXppc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXTIntel`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXTppc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXTsparc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXT68k`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`platformMacOSx86`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Register Component Resource flags

```
enum {
    registerComponentGlobal = 1,
    registerComponentNoDuplicates = 2,
    registerComponentAfterExisting = 4,
    registerComponentAliasesOnly = 8
};
```

Constants

`registerComponentGlobal`
Specify this flag to indicate that this component should be made available to other applications and clients as well as the one performing the registration. If you do not specify this flag, the component is available for use only by the registering application or component (that is, the component is local to the A5 world of the registering program).

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`registerComponentNoDuplicates`
Specify this flag to indicate that if a component with identical characteristics to the one being registered already exists, then the new one should not be registered (`RegisterComponent` returns 0 in this situation). If you do not specify this flag, the component is registered even if a component with identical characteristics to the one being registered already exists.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`registerComponentAfterExisting`
Specify this flag to indicate that this component should be registered after all other components with the same component type. Usually components are registered before others with identical descriptions; specifying this flag overrides that behavior.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`registerComponentAliasesOnly`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion

The functions `RegisterComponent` (page 38) , `RegisterComponentResource` (page 41) , and `RegisterComponentResourceFile` (page 42) use these flags in the `global` parameter.

Request Codes

```
enum {
    kComponentOpenSelect = -1,
    kComponentCloseSelect = -2,
    kComponentCanDoSelect = -3,
    kComponentVersionSelect = -4,
    kComponentRegisterSelect = -5,
    kComponentTargetSelect = -6,
    kComponentUnregisterSelect = -7,
    kComponentGetMPWorkFunctionSelect = -8,
    kComponentExecuteWiredActionSelect = -9,
    kComponentGetPublicResourceSelect = -10
};
```

Constants

`kComponentOpenSelect`

A request to open a connection. Your component must respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentCloseSelect`

A request to close a connection. Your component must respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentCanDoSelect`

A request to determine whether your component supports a particular request. Your component must respond to this request code

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentVersionSelect`

A request to return your component's version number. Your component must respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentRegisterSelect`

A request to determine whether your component can operate in the current environment. Your component may or may not respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentTargetSelect`

A request to call another component whenever your component would call itself. Your component may or may not respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentUnregisterSelect`
A request to perform any operations necessary as a result of your component being unregistered. Your component may or may not respond to this request code

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentGetMPWorkFunctionSelect`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentExecuteWiredActionSelect`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentGetPublicResourceSelect`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion
These values are used in the `ComponentParameters` (page 54) structure to specify the type of a request to a component. Apple has defined these request codes:

Set Default Component Flags

```
enum {
    defaultComponentIdentical = 0,
    defaultComponentAnyFlags = 1,
    defaultComponentAnyManufacturer = 2,
    defaultComponentAnySubType = 4,
    defaultComponentAnyFlagsAnyManufacturer = (defaultComponentAnyFlags  +
defaultComponentAnyManufacturer),
    defaultComponentAnyFlagsAnyManufacturerAnySubType =  (defaultComponentAnyFlags
 + defaultComponentAnyManufacturer +  defaultComponentAnySubType)
};
```

Constants
`defaultComponentIdentical`
The Component Manager places the component specified in the call to `SetDefaultComponent` in front of all other components that have the same component description.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyFlags`
The Component Manager ignores the value of the `componentFlags` field during the reorder operation.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyManufacturer`
The Component Manager ignores the value of the `componentManufacturer` field during the reorder operation.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnySubType`
The Component Manager ignores the value of the `componentSubType` field during the reorder operation.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyFlagsAnyManufacturer`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyFlagsAnyManufacturerAnySubType`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion
The `SetDefaultComponent` (page 45) function uses these values in the `flags` parameter to control which component description fields the Component Manager examines during the reorder operation.

Result Codes

The result codes defined by the Component Manager are listed below.

Result Code	Value	Description
`invalidComponentID`	-3000	Invalid component ID. Available in Mac OS X v10.0 and later.
`validInstancesExist`	-3001	This component has open connections. Available in Mac OS X v10.0 and later.
`componentNotCaptured`	-3002	This component has not been captured. Available in Mac OS X v10.0 and later.
`componentDontRegister`	-3003	Available in Mac OS X v10.0 and later.
`unresolvedComponentDLLErr`	-3004	Available in Mac OS X v10.0 and later.
`retryComponentRegistrationErr`	-3005	Available in Mac OS X v10.0 and later.
`badComponentSelector`	0x80008002	Component does not support the specified request code. Available in Mac OS X v10.0 and later.
`badComponentInstance`	0x80008001	Invalid component passed to Component Manager. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Component Manager selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Deprecated Component Manager Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.5

ComponentFunctionImplemented

Allows your application to determine whether a component supports a specified request. (Deprecated in Mac OS X v10.5.)

```
ComponentResult ComponentFunctionImplemented (
   ComponentInstance ci,
   SInt16 ftnNumber
);
```

Parameters

ci

The component instance of which you wish to make a request. Your application obtains the component instance from the `OpenDefaultComponent` (page 38) function or the `OpenComponent` (page 36) function. You can use a component identifier here, but you must coerce the data type appropriately.

ftnNumber

A request code value. See the documentation supplied with the component for request code values.

Return Value

Indicates whether the component supports the specified request. You can interpret this number as if it were a Boolean value. If the returned value is `TRUE`, the component supports the specified request. If the returned value is `FALSE`, the component does not support the request. Your application can use this function to determine a component's capabilities.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Components.h`

ComponentSetTarget

Calls a component's target request function and informs a component that it has been targeted by another component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult ComponentSetTarget (
   ComponentInstance ci,
   ComponentInstance target
);
```

Parameters

ci

The component instance to which to send a target request (the component that has been targeted). You can use a component identifier here, but you must coerce the data type appropriately.

target

The component instance issuing the target request.

Return Value

The value that the targeted component instance returns in response to the target request, or `badComponentSelector` if the targeted component does not support the target request.

Discussion

Your component can target a component instance without capturing the component or your component can first capture the component and then target a specific instance of the component.

You should not target a component instance if the component does not support the target request. Before calling this function, you should issue a can do request to the component instance you want to target to verify that the component supports the target request. After receiving a target request, the targeted component instance should call the component instance that targeted it whenever the targeted component instance would normally call one of its defined functions.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Components.h`

GetComponentVersion

Returns the version number of a component to your application. (Deprecated in Mac OS X v10.5.)

```
ComponentResult GetComponentVersion (
   ComponentInstance ci
);
```

Parameters

ci

The component instance from which you want to retrieve version information. Your application obtains the component instance from the `OpenDefaultComponent` (page 38) function or the `OpenComponent` (page 36) function.

Return Value

The version number of the component you specify. The high-order 16 bits represent the major version, and the low-order 16 bits represent the minor version. The major version specifies the component specification level the minor version specifies a particular implementation's version number.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Components.h`

RegisterComponentFile

(Deprecated in Mac OS X v10.5.)

```
OSErr RegisterComponentFile (
   const FSSpec *spec,
   short global
);
```

Parameters

spec

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Components.h`

RegisterComponentFileEntries

(Deprecated in Mac OS X v10.5.)

```
OSErr RegisterComponentFileEntries (
   const FSSpec *spec,
   short global,
   const ComponentDescription *toRegister,
   UInt32 registerCount
);
```

Parameters

spec
toRegister
registerCount

Return Value
A result code. See "Component Manager Result Codes" (page 69).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Components.h`

Document Revision History

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

Date	Notes
2006-07-17	Updated for Mac OS X v10.5. Added descriptions of thread safety mode functions.
2003-02-14	Updated formatting and linking.

Document Revision History

Index

B

C

D

E

F

G

I

K

M

N

O

P

R

S

U

V