Component Manager for QuickTime

QuickTime

2005-04-08

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

Simultaneously published in the United States and Canada.

Contents

Listings

Introduction to Component Manager for QuickTime

Many QuickTime services, such as image compression and decompression, are provided by components. Components are a type of shared code resource that you can manipulate using the Component Manager.

For the most part, QuickTime components are opened, configured, and closed as needed by QuickTime, without you as an application programmer having to work with them explicitly.

You will probably need to use the Component Manager from time to time, however, in order to open a specific component, to determine whether a component has specific properties, to modify the default configuration of a component, or to configure a component programmatically instead of invoking a user dialog.

Organization of this Document

This document describes the parts of the Component Manager you are likely to use in a QuickTime application. It includes a discussion of "Component Resources" (page 15), including component public resources, that can often be used to obtain information about components quickly, without actually opening the components.

This document also describes "Component Property Functions and Selectors" (page 14), introduced in QuickTime 6.4, that can be used to get and set component properties, and to install callbacks that are activated when component properties change, without using the Component Manager directly.

See Also

In addition to this document, you may find the following documents useful:

- For a high-level introduction to QuickTime components, see *QuickTime Overview* and the QuickTime QuickStart Guide.
- In addition to the conceptual information in this document, you should refer to the *Component Manager Reference*.
- If you intend to create new components, you also need to read Creating QuickTime Components.

Component Manager for QuickTime

The Component Manager is used to open, communicate with, and close QuickTime components such as image compressors and movie exporters. QuickTime typically opens, configures, and closes components automatically as needed. The Component Manager provides direct access to components for situations where QuickTime's default behavior is not sufficient.

Another way to find a component of a particular type or inspect the capabilities of components, without actually opening components and interrogating them, is to use the component properties API or the component public resources.

Component Manager Overview

The Component Manager creates an interface between components and clients. Clients can be applications, other components, system extensions, and so on. The Component Manager provides a standard interface your application can use to communicate with all components of a given type. Those components, in turn, provide the appropriate services to your client application.

All components of a given type share a common application interface and basic services. For example, all `'imdc'` components provide image decompression services using a common interface.

Individual components may also support additions to the defined application interface, but they must support the defined interface functions. Algorithm-dependent or implementation-dependent variations are implemented by components as extensions to the basic interface. For example, a decompressor component may support a unique compression technique or take advantage of special hardware, but all decompressor components support the `GetCompressedImageSize` function.

The Component Manager defines three kinds of component functions: functions called by client applications, functions component authors need to implement to allow system access, and support functions provided by the system to simplify the process of creating components. This document focuses on functions used by client applications.

The Component Manager allows applications to find components of a particular kind, such as image compressors, and to choose within available components of a kind, such as a JPEG or Cinepak image compressor.

Components are identified primarily by **type** and **subtype.** Both type and subtype are four-character codes: 32-bit values that are usually best interpreted as a sequence of four ASCII characters.

For example, an image decompressor for JPEG images has type `'imdc'` and subtype `'jpeg'`. A component's capabilities and characteristics can be further identified using the manufacturer's code (also a four-character code, such as `'aapl'`), or by getting the component's properties, or by interrogating the component directly.

Using the Component Manager, you can find out how many components of a specific type are available and get further details about a component's capabilities, without having to open the components. The Component Manager keeps track of many component characteristics, such as a component's name, icon, and information string.

The Component Manager allows a single component to serve multiple client applications at the same time. Each client application has a unique access path to the component. These access paths are called **component connections.** You identify a component connection by specifying a **component instance.** The Component Manager provides this component instance to your application when you open a connection to a component. The component maintains separate status information for each open connection.

For example, multiple applications might each open a connection to an image decompression component. The Component Manager routes each application request to the component instance for that connection. Because a component can maintain separate storage for each connection, application requests do not interfere with each other and each application has full access to the services provided by the component.

A function that finds a component typically returns a component identifier, referred to simply as a component. A function that opens a connection to a component returns a connection identifier, also known as a component instance.

Most QuickTime functions that take a component instance as a parameter can accept a component, as a convenience to the programmer, because the instance can typically be inferred.

This situation becomes ambiguous in threaded programming models, however. If you are calling QuickTime from more than one thread, be careful to always distinguish the component from the component instance.

Using the Component Manager

This section describes how you can use the Component Manager to:

- Gain access to components
- Locate components and take advantage of their services
- Get information about a component
- Close a connection to a component

Opening a Connection to a Default Component

You can allow the Component Manager to locate a suitable component for you. If you are interested in using a component of a particular type and subtype, and you do not need to specify any other characteristics of the component, use the `OpenADefaultComponent` function. This technique is the easiest way to open a component connection.

Note: This function replaces `OpenDefaultComponent`. The older function returns the component instance. The new function passes the component instance as a parameter and returns an `OSErr` value.

The `OpenADefaultComponent` function searches for available components and attempts to open a connection to a component with the specified type and subtype. If more than one component of the specified type and subtype is available, `OpenADefaultComponent` selects the first one in the list. If successful, it passes back a component instance that identifies your connection to the component.

Finding and Opening a Specific Component

To open a connection to a component based on information more specific than the component type and subtype, you can set up a component description structure (`ComponentDescription`) and call `FindNextComponent`, passing in `0` as the component identifier. `FindNextComponent` returns the first component it finds that matches the component description structure. Open the component using OpenAComponent.

Note: The newer function `OpenAComponent` replaces the older `OpenComponent`. The newer function passes the component instance back to the caller as a parameter and returns an `OSErr` value.

Getting Information About Components

If you need to choose among several components that match the component description structure, you can use `GetComponentInfo` to retrieve more information on each component. Follow these steps:

1. Set up a component description structure that describes the component.
2. Use the `CountComponents` function to determine how many components, if any, match your description.
3. Use the `FindNextComponent` function to iterate through the list, passing in `0` for the first component and the component ID of the previous component for each subsequent component, as shown in Listing 1.
4. Call `GetComponentInfo` to obtain further details on each component.
5. Open the component you want using the OpenAComponent function.

Listing 1-1 Iterating through a list of components

```
/*
   An application-defined function, myImportFinder, fills out a
   component description record to specify the search criteria for the
   desired component, then calls FindNextComponent to find a component
   with the specified characteristics--in this example, a movie
   importer component (any component with the type MovieImportType)
   that can open a particular file type  (subtype = someFileType).

   Begin by setting componentID to 0 and calling this function,
   passing in the desired file type to import. The function returns 0
   if a component is found, and componentID is set to the
   component ID. Call this function again without modifying
```

```
   omponentID to search for another matching component. The function
   returns 0 if another matching component is found. Repeat until you
   find the particular component that you want or until the function
   return is nonzero (no more matching components).

*/

long componentID

pascal Boolean myImportFinder(OSType someFileType)

{
  ComponentDescription cd;
  Boolean result = true;      // true means no matching importer

  cd.componentType = MovieImportType;
  cd.componentSubType = someFileType;
  cd.componentManufacturer = 0; //any manufacturer
  cd.componentFlags = canMovieImportFiles;
  cd.componentFlagsMask = canMovieImportFiles;
  componentID = (FindNextComponent(componentID, &cd))  // search for component
 to do the work
  if componentID
    result = false;

  return result;
}
```

Alternatively, you can use the component property functions to get the properties of components, though not all components support this. For more information, see "Component Property Functions and Selectors" (page 14).

Some components also provide information about themselves in public resources. For details, see "Component Resources" (page 15).

Calling Component Functions

The specific component functions available to applications vary considerably with the component type. See the documentation for a given type of component to see what functions it provides that are unique.

The Component Manager defines a number of functions that all components should support, however, regardless of component type. For example, all components should support `GetComponentInfo` and `GetComponentVersion`. In addition, you can ask any component if it supports a given function by calling ComponentFunctionImplemented .

For more information, see "Component Manager Functions and Data Structures for Applications" (page 13).

Closing Component Connections

When your application is done with a component, close the connection by calling `CloseComponent`. When all component instances are closed, the component is released.

Component Manager Functions and Data Structures for Applications

These are the Component Manager functions and data types commonly used by applications to work with components.

Additional functions that may be available to applications vary with the component type. See the documentation for a specific component to see what it does.

The functions listed here should be supported by all components, regardless of type. In addition, you can ask any component if it supports a given function by calling `ComponentFunctionImplemented`.

For a complete list of Component Manager functions and data types, see *Component Manager Reference*.

Finding components:

- `FindNextComponent`
- `CountComponents`
- `GetComponentListModSeed`
- `GetComponentTypeModSeed`

Opening and closing components:

- `OpenADefaultComponent`
- `OpenAComponent`
- `CloseComponent`

Getting information about components:

- `GetComponentInfo`
- `GetComponentIconSuite`
- `GetComponentVersion`
- `ComponentFunctionImplemented`

Retrieving component errors:

- `GetComponentInstanceError`

Data types used by applications:

- `ComponentDescription`
- `ComponentInstanceRecord`
- `ComponentRecord`

Component Property Functions and Selectors

Component property functions, which are available in QuickTime 6.4 and later, provide another way to get information about components. They can also be used to configure components without a user dialog, to configure a component programmatically, and to set up callbacks that are called when something changes in a component.

The configuration interface to many components, such as those used for capture, export, and compression, involves a standard user dialog. QuickTime-based applications sometimes need to configure these operations from custom user interfaces or to configure them programmatically, with no user interaction at all.

The following functions are available to work with component properties:

- `QTGetComponentPropertyInfo` gets information about a specified property of a component, such as the size and data format.
- `QTGetComponentProperty` gets the value of a specific component property.
- `QTSetComponentProperty` sets the value of a specific component property programmatically.
- `QTAddComponentPropertyListener` installs a callback to monitor a component property. Your callback function is called when the property value changes.
- `QTRemoveComponentPropertyListener` removes a component property monitoring callback.
- `QTComponentPropertyListenerCollectionCreate` creates a collection of component property monitors, allowing you to manage groups of callback functions.
- `QTComponentPropertyListenerCollectionAddListener` adds a listener callback for a specified property class and ID to a property listener collection.
- `QTComponentPropertyListenerCollectionRemoveListener` removes a listener callback with a specified property class and ID from a property listener collection.
- `QTComponentPropertyListenerCollectionNotifyListeners` calls all listener callbacks in a component property listener collection registered for a specified property class and ID.
- `QTComponentPropertyListenerCollectionHasListenersForProperty` determines if there are any listeners in a component property listener collection registered for a specified property class and ID.
- `QTComponentPropertyListenerCollectionIsEmpty` determines if a listener collection is empty.

Beginning with Quicktime 6.4, the Component Manager defines optional standard selectors for component properties. These selectors can be implemented by all components, regardless of type, that are written to work with the component properties mechanism. Components can adopt these functions on their own schedule. The Component Manager allows callers to discover safely whether components implement the property selectors and perform operations according to older mechanisms if support for the properties mechanism is unavailable.

As developers adopt the component property functions, it will be possible to configure most or all exporters, for example, by means of the same set of properties. Use of the component property functions allows support for richer scripting and simpler custom user interfaces.

Component Resources

Component resources provide an alternate method of gathering detailed information about components, without opening them and interrogating them. This can be a great time saver when you need to get specific information about a lot of components.

Macintosh programmers should be familiar with the idea of resources, commonly a collection of static data bundled with an application, such as the text strings used in menus and the icon that represents the file. This bundling provides a convenient way to separate data that may be localized to a language, for example, from program code that can be modified only with extreme care.

Resources can be stored inside the file they belong to or in a parallel file, typically a file in the same directory and having the same name, but with a different filename extension. QuickTime provides support for both kinds of resource storage on Windows operating systems. In Mac OS X, resource files are not normally visible; they are bundled inside a folder represented by the parent file's icon.

QuickTime components can contain resources, which are simply pieces of data. Component resources are typically used as a way for components to store data that is accessed directly only by the component itself. QuickTime 4 and later also supports public resources, which can be accessed by the system and by application programs. Of course, a component can access its own public resources as well.

A component's resources are identified by an `OSType` value and an ID. A private resource's `OSType` value and ID are the same as the Mac OS resource type and ID that the resource is identified with in the component's resource file.

A component's public resources are also identified by an `OSType` value and an ID, but they do not have to be the same as the Mac OS resource type and ID that the resource is stored in.

A component that provides public component resources must provide a component resource map, which indicates the mapping between the public resource `OSType` and ID and the private `OSType` and ID. Listing 2 shows an example of a component resource map.

Listing 1-2 Example of a component resource map

```
resource 'thnr' (512) {
        {
                'PICT', 1, 0,
                'pict', 128, 0,

                'PICT', 2, 0,
                'pict', 129, 0,
        }
}
```

This component resource map makes available two public resources, `'PICT' 1` and `'PICT' 2`. The map shows that these two public resources are stored in the component as Mac OS resources `'pict' 128` and 'pict' `129`.

An application can obtain a public resource from a component by using the `GetComponentPublicResource` function, as shown in the following code snippet:

```
err = GetComponentPublicResource (
   (Component)store->self, 'PICT' , 1, &resource);
```

For each call to `GetComponentPublicResource`, the Component Manager may have to open the component's resource file and load a resource. Resource caching makes this fast for repeated calls to the same resource, but if many components are to be accessed in turn, the first pass may be quite slow.

Often, however, multiple components of the same type are stored in a single file. QuickTime itself contains nearly a dozen movie exporters, for example. Storing components of the same type in one file makes significant optimization possible; to support this, QuickTime provides the function `GetComponentPublicResourceList`.

`GetComponentPublicResourceList` is a single optimized call to obtain a specified public resource from each component that matches a component description. Listing 3 is an example that shows how to get all the `'PICT' 1` public resources from all installed movie exporters. The resources are passed back as an atom container, with each resource in its own atom.

Listing 1-3 Getting a resource from all components of a type

```
QTAtomContainer resources;
ComponentDescription cd;

cd.componentType = MovieExportType;
cd.componentSubType = 0;
cd.componentManufacturer = 0;
cd.componentFlags = 0;
cd.componentFlagsMask = 0;

err = GetComponentPublicResourceList ('PICT', 1, 0, &cd, nil, nil, &resources)
```

Once the call to `GetComponentPublicResourceList` has completed, the QT atom container returned contains all the `'PICT' 1` public resources for all movie exporters installed. Listing 4 shows how to access these resources.

Listing 1-4 Accessing resources from a component resource list

```
long i, count;

count = QTCountChildrenOfType(resources,
kParentAtomIsContainer, OSTypeConst('comp'));
for (i=1; i<=count; i++) {
        QTAtom compAtom, resourceAtom;
        Component c;

        compAtom = QTFindChildByIndex(
        resources, kParentAtomIsContainer,
        OSTypeConst('comp'), i, (long *)&c);
        resourceAtom = QTFindChildByID(
         resources, compAtom, 'PICT', 1, nil);
        if (resourceAtom) {
                Handle resource = NewHandle(0);

                QTCopyAtomDataToHandle(
        resources, resourceAtom, resource);

                // do something with the resource for Component c

                DisposeHandle(resource);
        }
```

```
}

QTDisposeAtomContainer(resources);
```

Notice that the resource atom's ID is returned as the component atom's `compAtom` reference.

Although using `GetComponentPublicResourceList` can take more code, it is often substantially faster than repeated calls to `GetComponentPublicResource`. You can use either approach, but using `GetComponentPublicResourceList` is recommended as its performance is likely to remain high, especially as more components are added to QuickTime.

Document Revision History

This table describes the changes to *Component Manager for QuickTime*.

Date	Notes
2005-04-08	Corrected revision history and date.
2004-12-02	Rewritten document that shows how to work with QuickTime components using the Component Manager, the component properties API, and component resources.

Document Revision History