64-Bit Guide for Carbon Developers

Carbon > Performance

2007-12-11

Contents

Chapter 4   **Changes in Other C APIs   27**

Introduction to 64-Bit Guide for Carbon Developers

One of Apple's goals for Mac OS X version 10.5 (Leopard) is to make it possible for applications to use 64-bit addressing. System libraries and frameworks are now 64-bit ready, meaning they can be used in both 32-bit and 64-bit applications. With this support, you can create applications that address extremely large data sets. On Intel-based Macintosh computers, some 64-bit applications may even run faster than their 32-bit equivalents because of the availability of extra processor resources in 64-bit mode.

Most APIs in Mac OS X v10.5 are available to both 32-bit and 64-bit applications, but some APIs commonly used by Carbon applications are not. In particular, the APIs used to implement a Carbon user interface are generally available only to 32-bit applications. If you want to create a 64-bit application for Mac OS X, you need to use Cocoa to implement its user interface.

You may not need to develop 64-bit versions of your applications right now, but you can start to prepare your projects for this transition. As part of this process, you need to adopt standard data types and implement some application features with alternative technologies. This document discusses guidelines, issues, and procedures specific to the transition to 64-bit executables for Carbon developers.

Who Should Read This Document?

This document is recommended reading for Carbon developers who want to learn what changes are necessary to create 64-bit executable versions of their applications.

Organization of This Document

This document contains the following chapters:

See Also

This document is a supplement to *64-Bit Transition Guide*, the definitive guide to 64-bit computing in Mac OS X. If you haven't already done so, you should read *64-Bit Transition Guide* before reading this document.

For Cocoa-specific information, you should read *64-Bit Transition Guide for Cocoa*.

You may also want to read *Tiger Developer Overview Series: Developing 64-bit Applications*, an ADC featured article that discusses 64-bit computing in Mac OS X v10.4.

The Transition to 64-Bit Addressing

This chapter describes the overall 64-bit initiative for Mac OS X and offers advice and guidelines for moving your projects to 64-bit addressing.

> **Note:** This chapter summarizes some of the background information, requirements, and issues related to 64-bit executables presented in *64-Bit Transition Guide*. Refer to that document for a more complete description.

The 64-Bit Initiative

Since version 10.4 (Tiger), Mac OS X has been moving to a model that supports a 64-bit address space. In this model, called LP64, both `long` integers and pointers are 8 bytes (64 bits) instead of 4 bytes. In addition, the `size_t` integer type is 8 bytes instead of 4 bytes. The alignment of these types for LP64 has also increased to 8 bytes. The sizes of all other primitive integer types (`char`, `int`, `off_t`, and so on) remain as they are in the 32-bit model (ILP32), but the alignment of some—namely `long long` and `pos_t`—has increased to 8 bytes for LP64.

On Intel architectures, 64-bit mode also entails an increase both in the number of registers and in their width, as well as a change in the calling conventions to pass arguments in registers instead of on the stack. As a direct consequence of these changes, 64-bit executables running on Intel-based Macintosh computers may see a boost in performance. (The expansion of registers does not happen on the PowerPC architecture, because it was designed for 64-bit computing from the outset.) Although the kernel remains 32-bit in Mac OS X v10.5, it supports 64-bit software in user space.

All pointers in a 64-bit process are 64 bits—there is no "mixed mode" in which some pointers are 32 bits and others are 64 bits. Consequently, all supporting binaries needed to run a process, including frameworks, libraries, and plug-ins, must be 64-bit capable if the process is to run in a 64-bit address space. All dependencies require porting to 64 bit.

As part of the 64-bit initiative for Mac OS X v10.5 (Leopard), Apple is porting system frameworks, libraries, and plug-ins to support 64-bit addressing. They are packaged to support both 32-bit and 64-bit executables. Thus, if you have a 32-bit application and a 64-bit application running at the same time, both framework stacks on which the applications have dependencies are loaded into memory. The GCC compiler, linker, debugger, and other development tools have also been modified to support 64-bit addressing. System daemons are also being modified to support 64-bit processes. The Cocoa frameworks as well as the Objective-C runtime and related development tools are part of the porting effort. Frameworks with procedural C APIs support 64-bit executables as well, although a number of Carbon managers, system services, and individual functions are not available.

Several changes have also been made to intermediate integer types in lower layers of the system. For example, the underlying primitive type for `CFIndex` in Core Foundation has been changed to a 64-bit type, while the underlying type for `SInt32` in Carbon Core has been changed to remain 32 bits in a 64-bit world. These changes percolate up into higher layers of the system, affecting frameworks where they expose these types in their APIs.

There are several consequences and implications of a transition to a 64-bit address space:

- A 64-bit executable can concurrently access 16 exabytes of virtual address space.

- System memory requirements will more than double, because data structures are larger and two framework stacks are loaded simultaneously.

An important stake of the 64-bit initiative is to maintain binary compatibility for existing 32-bit applications after 64-bit changes are made to system frameworks, libraries, and plug-ins. Sometimes this means keeping the underlying primitive type the same for 32-bit values.

When and How Should I Move My Project to 64 Bit?

For Mac OS X v10.5 (Leopard), just a small percentage of projects have any need to move to a 64-bit address space. Generally, these projects are for applications that require the increased address space for manipulating large data sets, or that need to have random access to data objects exceeding 4 GB. Some examples of applications that fall into this category include those that perform scientific computing, large-scale 3D rendering and animation, data mining, and specialized image processing.

> **Note:** If you have a project that builds a publicly available framework, library, or plug-in, you should port it to 64 bit for Mac OS X v10.5. This is especially true if your framework, library, or plug-in has an API that needs to cover a range of values that cannot be handled with 32 bits. Similarly, if you own a server or daemon that needs to talk with processes that can be either 32 bit or 64 bit, you need to make sure your IPC mechanism can deal with both 32-bit and 64-bit pointers and data structures.

For releases after Mac OS X v10.5, Apple expects that more and more software projects—including most consumer applications—will make the transition to 64 bit. There are several reasons for this expectation:

- **Competing platforms**. Microsoft Windows XP already has a 64-bit version, and Windows Vista is also 64-bit capable.

- **Hardware evolution**. As the downward trend in the price of hardware components (such as memory chips) continues, the typical configuration of computers will grow to allow 64-bit computing to become the norm. Moreover, as noted above, you may see performance improvements for 64-bit executables running on Intel-based Macintosh computers.

- **User demand**. Along with hardware configurations, user expectations will also grow. On a 64-bit-capable Mac OS X system with, say, 32 GB of memory, many users will be unhappy with an application that can access no more than 4 GB of data.

So although most applications don't immediately need to make the transition to 64-bit addressing, in a few years' time they will. You can start now to prepare your projects for this transition, proceeding in stages.

1. Begin adopting Mac OS X APIs that are available to 64-bit applications. Modify your own interfaces and code to take advantage of a 64-bit address space and 64-bit quantities. Add preprocessor conditionals as needed to maintain source compatibility.

2. Fix any assignments that result in pointer or long integer truncation in 64-bit mode. Ensure identical size and alignment for data structures that are shared between 32-bit and 64-bit code (across a network or in a file, for example).

3. Test your project to ensure that all code paths are capable of dealing with pointers and quantities that take on values greater than 2^32.

You can use existing technologies to make your application "universal" by packaging binaries in your application bundle with 64-bit and 32-bit variants combined with variants for PowerPC and Intel architectures. Thus your application could have four-way multi-architecture binaries, with binaries for 32-bit PowerPC, 64-bit PowerPC, 32-bit Intel, and 64-bit Intel. Generally, you will want a 64-bit application to ship with 32-bit binaries as well, since you want it to run on 32-bit-only hardware, which will be common for many years.

Modifying Your Application to Use 64-Bit Addressing

This chapter explains what kinds of modifications are needed in existing Carbon applications to make them compile and run as 64-bit executables.

Mac OS X v10.5 (Leopard) provides two different implementations of its programming interfaces: one for 32-bit applications and another for 64-bit applications. You may choose to build your application using either one or both of these implementations. Some Leopard header files have been modified to support both implementations, and these header files often select or remove code for 64-bit compilation.

For 32-bit applications, Apple's goal is to maintain complete source compatibility; when recompiled on Leopard, 32-bit applications should not need to change their sources to compile against Leopard headers. If you find instances where changes to the Leopard headers have broken the compilation of your 32-bit application, you should treat these as bugs and report them to Apple.

If you plan to modify your application to use 64-bit addressing, you may need to adopt standard data types that can be used identically in both 32-bit and 64-bit applications. You also need to be aware that the APIs used to implement a Carbon user interface—menus, windows, views, toolbars, navigation dialogs, and so on—are generally not available. If you want to create a 64-bit application, you need to use Cocoa to implement its user interface.

Later in this document, you'll learn more about replacement alternatives for the APIs and technologies that aren't available to 64-bit applications.

Conditional Compilation

The differences between the 32-bit and 64-bit programming interfaces are marked in header files using the following preprocessor conditionals:

```
#if __LP64__
(code available only to 64-bit applications)
#endif

#if !__LP64__
(code available only to 32-bit applications)
#endif
```

These two conditionals are used to select or remove code for 64-bit compilation in a manner that is independent of the machine architecture (PowerPC or Intel). This is the primary mechanism used to provide 32-bit and 64-bit versions of an API or a standard data type such as `CGFloat`.

Standard Types

In the context of this document, standard types are Apple-defined data types that can be used identically in both 32-bit and 64-bit applications. In Mac OS X v10.5, Apple has redefined some of these types (or defined new types) to make it easier for you to maintain a source base that compiles for both 32-bit and 64-bit targets. These standard types are used in function parameters and return values to implement 64-bit addressing and 64-bit quantities when compiling 64-bit targets. You may need to modify your source code to take full advantage of these wider parameters and return values.

Changes in Standard Types

Commonly used standard types such as `ByteCount`, `ByteOffset`, `ItemCount`, and `UniCharCount` have been defined as `long`, which makes them 64-bit types for 64-bit applications:

```
typedef unsigned long     ByteCount;
typedef unsigned long     ByteOffset;
typedef unsigned long     ItemCount;
typedef unsigned long     UniCharCount;
```

New standard types have been added to provide consistent representation of user-specified data:

```
typedef void *            PRefCon;
#if __LP64__
typedef void *            URefCon;
typedef void *            SRefCon;
#else
typedef UInt32            URefCon;
typedef SInt32            SRefCon;
#endif
```

Core Foundation has redefined several standard types to accommodate wide values for 64-bit applications:

```
typedef unsigned long     CFTypeID;
typedef unsigned long     CFOptionFlags;
typedef unsigned long     CFHashCode;
typedef signed long       CFIndex;
```

Core Graphics (Quartz) has defined a new standard type to increase the precision of floating-point values for 64-bit applications:

```
#if __LP64__
typedef double CGFloat;
#else
typedef float CGFloat;
#endif
```

It's also worth noting that declarations of the fixed-size types `SInt32` and `UInt32` have been changed to make them remain 32-bit quantities for 64-bit applications:

```
#if __LP64__
typedef unsigned int      UInt32;
typedef signed int        SInt32;
#else
typedef unsigned long     UInt32;
```

```
typedef signed long        SInt32;
#endif
```

Adopting Standard Types

The following general changes have been made in a number of C APIs to adopt standard types. These changes should have no effect on your application.

- Many functions and data structures now use standard types, such as `ItemCount`, `ByteCount`, `ByteOffset`, and `UniCharCount`, rather than fixed-size types, such as `UInt32` and `SInt32`.

- Function parameters that contain user-specified data and that were previously typed as a 32-bit integer value (`SInt32` or `UInt32`) are now typed as `SRefCon` or `URefCon`. For 64-bit applications, these new standard types will widen as necessary to allow a pointer to be passed in and out.

- Graphics-related functions that formerly used the `float` type now use the `CGFloat` type defined by Core Graphics. This type expands to `double` for 64-bit applications.

To advance your application toward the goal of being 64-bit capable, you are encouraged to adopt these standard types in your own source code. At a minimum, you need to make sure that the data you pass to and receive from system functions uses compatible types.

For example, if you are currently casting a pointer to `SInt32` or `UInt32` before passing it as user-specified data, you should instead cast the pointer to `SRefCon` or `URefCon` to avoid truncating your pointer in 64-bit targets. If you are currently passing an integer value as user-specified data, you will need to add a cast to `SRefCon` or `URefCon` to avoid errors when compiling your code for 64-bit targets.

Common Pitfalls

If you're already using standard types, your code may be making assumptions about their sizes that no longer hold true in 64-bit applications. Here are some examples.

Example 1

In this example, the code assumes that `ItemCount` is a 32-bit type.

```
ItemCount count;   // Used to be declared as UInt32, now unsigned long
memmove(mem, &count, 4);
```

Instead of `4`, `sizeof(ItemCount)` should have been used.

Example 2

In this example, the code assumes that `CFIndex` is the same size as `SInt32`.

```
CFIndex c;   // Used to be declared as SInt32, now long
CFNumberRef n = CFNumberCreate(NULL, &c, kCFNumberSInt32Type);
```

Instead of `kCFNumberSInt32Type`, `kCFNumberCFIndexType` should have been used.

Example 3

In this example, the code assumes that a prototype exists for a function that returns a pointer-sized type.

```
CFStringRef s = SomeGetStringFunction();
```

If a prototype for `SomeGetStringFunction` is not encountered, the compiler assumes an `int`, hence 32-bit, return value.

Example 4

This example uses variable-sized data across process boundaries—in this case, by writing the data to a file.

```
/* 32-bit process */
CGFloat floatValue;
fwrite(&floatValue, sizeof(floatValue), 1, file);  // writes out 4 bytes

/* 64-bit process */
CGFloat floatValue;
fread(&floatValue, sizeof(floatValue), 1, file);  // tries to read back 8 bytes
```

Basically, any use of variable-size types in data structures that go outside the address space of a process should be examined.

Example 5

The Collection Manager is available to 64-bit applications, but unlike many other APIs, the `itemSize` parameters have not been widened to 64 bits; they remain `SInt32`. This can cause problems if your code is already casting an `itemSize` parameter to an `SInt32*` type before calling a Collection Manager function. For example, this code will compile without warnings in a 64-bit target but will fail at runtime because `sizeof(ByteCount)` is no longer equal to `sizeof(SInt32)`.

```
MyData data;
ByteCount itemSize = sizeof(data);
GetCollectionItem (collection, kDataTag, kDataID, (SInt32*) &itemSize, &data);
```

Passing the address of a `ByteCount` (or `long`) variable to `GetCollectionItem` causes this function to return zero bytes instead of `sizeof(data)` bytes. The same issue applies to `GetIndexedCollectionItem` and `GetTaggedCollectionItem`. Consider reviewing your use of these functions to verify that you are using the correct parameter type for the `itemSize` parameter.

Example 6

Some Core Foundation functions return a value by reference rather than as the function result. It's easy to pass the address of an incorrectly sized variable to one of these functions. For example, suppose you have stored an integer in a dictionary and you wish to retrieve its value.

```
UInt32 value = 0;
if (CFDictionaryGetValueIfPresent (dict, CFSTR("key"), (void**) &value))
    DoSomething (value);
```

In a 64-bit target, this code will compile without warnings but will fail at runtime because `sizeof(UInt32)` is no longer equal to `sizeof(void*)`. The `CFDictionaryGetValueIfPresent` function will write an 8-byte value into the 4-byte memory location that you have provided, overwriting some other memory. To fix this, use a variable of type `long`, `unsigned long`, `CFIndex`, or `intptr_t`, any of which will properly resize to match the size of a pointer.

What's Not Available to 64-Bit Applications

This section lists a few of the managers, services, and individual functions that are not available to 64-bit applications. You can find more detailed information about these changes in the next two chapters, "Changes in the Human Interface Toolbox" (page 19) and "Changes in Other C APIs" (page 27).

Deprecated Technologies

A number of legacy Carbon technologies have been either wholly or partially deprecated in favor of more modern equivalents. You are discouraged from using these technologies in 32-bit applications. If you are creating 64-bit applications, you must use alternative technologies.

For example, these deprecated technologies are not available to 64-bit applications:

■   QuickDraw. Functions to manipulate regions and simple data structures (`Point` and `Rect`) are still available, but you cannot use QuickDraw to draw in a 64-bit application. See "QuickDraw" (page 37).

■   The `FSSpec` data type and functions that use this type. Most of these functions have replacements that use the `FSRef` type. See "File Manager" (page 32).

■   Classic icon formats such as `'ICON'`, `'cicn'`, and icon suites. You must use the `IconRef` data type or a CGImage to represent an icon. See "Icon Services and Utilities" (page 33).

■   Memory Manager. A number of deprecated or obsolete functions are not available. See "Memory Manager" (page 35).

■   Scrap Manager. You can use the Pasteboard Manager or `NSPasteboard` instead. See "Scrap Manager" (page 25).

■   Display Manager. You must use the `CGDirectDisplayID` data type and Quartz Display Services instead. See "Display Manager" (page 32).

■   Sound Manager. You should use Core Audio instead. See "Sound Manager" (page 39).

Supported Technologies

Some other technologies are still supported for 32-bit applications but are not available to 64-bit applications. Among these technologies are the following:

■   Carbon Human Interface Toolbox. APIs such as the Window Manager, Menu Manager, Data Browser, HIView, HIToolbar, and HIArchive are not available to 64-bit applications. You must implement your user interface with Cocoa. See "Choosing a Development Path for Your Carbon User Interface" (page 18).

■   Navigation Services. You must use Cocoa to implement dialogs for navigating, opening, and saving files. See "Navigation Services" (page 25).

- The Carbon Printing API. You must use Cocoa to implement printing features that display a user interface. See "Printing" (page 35).

- QuickTime C APIs. You must use QuickTime Kit instead. See "QuickTime" (page 37).

- MLTE. You should use the Cocoa text system instead. See "Multilingual Text Engine (MLTE)" (page 25).

- ATSUI. You should use Core Text or Cocoa instead. See "Apple Type Services for Unicode Imaging (ATSUI)" (page 28).

- Carbon APIs for displaying standard color and font selection windows. You should use Cocoa to display these windows. See "Color Picker Manager" (page 29) and "Fonts Window Services" (page 33).

Choosing a Development Path for Your Carbon User Interface

During the evolution of Mac OS X, a number of improvements and new features have been added to Carbon to help you modernize your Carbon user interface (UI) and begin to incorporate Cocoa features. You have been encouraged to adopt newer UI technologies such as composited windows, HIView-based controls, and Quartz 2D for drawing. In Mac OS X v10.5, the addition of HICocoaView has opened up a number of possibilities for adding Cocoa features to applications that use Carbon windows.

Because most Carbon UI functions are not available to 64-bit applications, you have two possible development paths. You can continue modernizing and improving your Carbon UI with the expectation that your application will remain a 32-bit application for the foreseeable future. Apple plans to support and maintain the 32-bit Carbon Human Interface Toolbox, although Apple will not be adding any significant new features to these APIs. The other development path is more challenging and also potentially more rewarding in the long term. You can develop a 64-bit version of your application, using Cocoa to implement your UI. As you do so, consider going one step further and implementing other parts of your application using Cocoa. For an introduction to Cocoa programming, see *Cocoa Fundamentals Guide*.

At this time, a reasonable approach would be to compare the amount of work required for each development path, assess the potential benefits, and decide which option is more attractive.

Changes in the Human Interface Toolbox

This chapter discusses 64-bit changes in the Carbon Human Interface Toolbox, which includes APIs in the HIToolbox, HIServices, and Navigation Services frameworks. The APIs are listed in alphabetical order.

> **Note:** To find out if a specific function is not available to 64-bit applications, you can check the API reference documentation in the ADC Reference Library, use the Xcode 3.0 Research Assistant, or consult the relevant header file.

Appearance Manager

Carbon: Appearance.h

Most Appearance Manager functions are not available to 64-bit applications. The Appearance Manager is largely based on QuickDraw, while the newer HITheme API is based on Quartz 2D. You should use HITheme to draw appearance primitives. Refer to *Appearance Manager Reference* for detailed information about function availability.

The function `GetThemeMetric` is available so that 64-bit applications can determine:

- How large various theme-capable objects should be
- The size of the bounding rectangle to pass to HITheme functions to get the right appearance

Application Manager

Carbon: MacApplication.h

A number of functions in the Application Manager are not available to 64-bit applications. Refer to *Application Manager Reference* for detailed information about function availability.

Because there are no Cocoa equivalents, the following functions are still available:

- `GetSystemUIMode`
- `SetSystemUIMode`
- `GetApplicationTextEncoding`
- `HISearchWindowShow`
- `HIDictionaryWindowShow`

Carbon Event Manager

Carbon: CarbonEvents.h, CarbonEventsCore.h

Some Carbon Event Manager functions and events are not available to 64-bit applications. Refer to *Carbon Event Manager Reference* for detailed information about function availability. To learn about handling events in Cocoa, see *Cocoa Event-Handling Guide*.

Carbon Event Parameters

With the introduction of 64-bit support in the Carbon Event Manager, four kinds of event parameters require changes to their underlying data types. The parameters that have changed are used to pass user-specified data, byte counts, byte offsets, and graphics devices.

Refcon Data, Byte Counts, and Byte Offsets

In the case of refcon data, byte counts, and byte offsets, these parameters use `typeSInt32` or `typeLongInteger` as the event parameter type prior to Mac OS X v10.5. Unfortunately, `typeSInt32` and `typeLongInteger` specify a 32-bit value, and in Mac OS X v10.5, these parameters must be 64 bits wide when compiling for the 64-bit API so that 64-bit pointers, byte counts, and byte offsets may be passed without truncation.

The Carbon Event Manager defines three new event parameter types to provide source compatibility between 32-bit and 64-bit applications: `typeRefCon`, `typeByteCount`, and `typeByteOffset`. Use of these new types by 32-bit applications is optional. The toolbox provides automatic coercion between the old and new types so existing applications will continue to run. You may recompile your 32-bit application in Mac OS X v10.5 without modification, but in a 64-bit application you must modify your source to use the new types.

The actual value of these constants depends on whether you are compiling for a 32-bit or 64-bit target and on the value of the preprocessor macro `MAC_OS_X_VERSION_MIN_REQUIRED`, which specifies the minimum Mac OS X system version on which your application will run.

When compiling your code using the Mac OS X v10.5 header files, if you want your code to run cleanly in both 32-bit and 64-bit targets, you should modify your calls to `GetEventParameter` to use `typeRefCon`, `typeByteCount`, or `typeByteOffset` instead of `typeSInt32` or `typeLongInteger`. Using these preferred types will allow your 32-bit application to run in Mac OS X v10.5 and earlier, and will allow your 64-bit application to run in Mac OS X v10.5 and later.

Note that when you extract data from an event parameter of type `typeRefCon`, the data type of the variable to which the parameter data is written should always be pointer sized; in other words, use `SRefCon`, `URefCon`, or `PRefCon`, not `SInt32`. Using a 32-bit variable will fail on 64-bit targets. Similarly, when extracting data from an event parameter of type `typeByteCount` or `typeByteOffset`, always use the corresponding standard type (`ByteCount` or `ByteOffset`), and use `sizeof(ByteCount)` or `sizeof(ByteOffset)`, not `sizeof(SInt32)` or `sizeof(UInt32)`, to specify the amount of data that you need.

The following Carbon events use `typeRefCon`:

```
kEventTSMDocumentAccessGetLength
kEventTSMDocumentAccessGetSelectedRange
kEventTSMDocumentAccessGetCharactersPtr
kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer
```

```
kEventTSMDocumentAccessGetCharacters
kEventTSMDocumentAccessGetFont
kEventTSMDocumentAccessGetGlyphInfo
kEventTSMDocumentAccessLockDocument
kEventTSMDocumentAccessUnlockDocument
kEventTextInputUpdateActiveInputArea
kEventTextInputUnicodeForKeyEvent
kEventTextInputOffsetToPos
kEventTextInputPosToOffset
kEventTextInputShowHideBottomWindow
kEventTextInputGetSelectedText
kEventTextInputFilterText
kEventAppLaunchNotification
kEventControlArbitraryMessage
```

The following Carbon events use `typeByteCount`:

```
kEventTextInputUpdateActiveInputArea
kEventControlSetData
kEventControlGetData
```

The following Carbon events use `typeByteOffset`:

```
kEventTextInputOffsetToPos
kEventTextInputPosToOffset
```

Graphics Devices

In the case of graphics devices, some event parameters used `typeGDHandle` prior to Mac OS X v10.5. However, the `GDHandle` type is not available to 64-bit applications at runtime. In Mac OS X v10.5 and later, these parameters contain values of type `CGDirectDisplayID` instead.

The Carbon Event Manager defines a new event parameter type, `typeCGDisplayID`, to indicate that the data type of a graphics device event parameter is `CGDirectDisplayID` instead of `GDHandle`. You need to explicitly choose in your code the type of graphics device identifier you need. Use of `typeCGDisplayID` by 32-bit applications is optional. The toolbox provides automatic coercion between `typeCGDisplayID` and `typeGDHandle`, so existing applications will continue to run. You may recompile a 32-bit application in Mac OS X v10.5 without modification, but in a 64-bit application you must use `typeCGDisplayID` to retrieve the graphics device identifier.

The following 32-bit Carbon events use `typeCGDisplayID`:

```
kEventAppAvailableWindowBoundsChanged
kEventWindowConstrain
kEventMenuGetFrameBounds
```

Carbon Help Manager

Carbon: MacHelp.h

The Carbon Help Manager is used to display help tags for elements in an application's user interface. You typically define help tags in Interface Builder, where they are called tool tips.

The Carbon Help Manager is not available to 64-bit applications. To learn about displaying help tags for Cocoa views, see *Online Help*.

Control Manager

Carbon: Controls.h and related headers

The Control Manager is not available to 64-bit applications. In a Cocoa user interface, you must use Cocoa controls. See *Control and Cell Programming Topics for Cocoa*.

Data Browser

Carbon: HIDataBrowser.h

The Data Browser is not available to 64-bit applications. The recommended replacements are the Cocoa classes `NSBrowser`, `NSTableView`, and `NSOutlineView`. For additional information about Cocoa browsers, see *Browsers*.

Dialog Manager

Carbon: Dialogs.h

The Dialog Manager is not available to 64-bit applications. Cocoa provides both sheet and application-model dialogs. For information about using dialogs or panels in Cocoa, see *Window Programming Guide for Cocoa* and *Sheet Programming Topics for Cocoa*.

Drag Manager

Carbon: Drag.h

The Drag Manager is not available to 64-bit applications. For information about drag-and-drop capabilities in Cocoa windows and views, see *Drag and Drop Programming Topics for Cocoa*.

Event Manager

Carbon: Events.h

Most Event Manager functions are not available to 64-bit applications. Refer to *Event Manager Reference* for detailed information about function availability. To learn about handling events in Cocoa, see *Cocoa Event-Handling Guide*.

The `GetMouse` function returns a point in local coordinates in the current graphics port. Since there is no QuickDraw graphics port in 64-bit applications, this function is not available. You should use the Carbon Event Manager function `HIGetMousePosition` instead. See `Carbon/HIToolbox/CarbonEventsCore.h`.

The `KeyTranslate` function is not available. You should use `UCKeyTranslate` in Unicode Utilities instead.

HIArchive

Carbon: HIArchive.h

HIArchive is not available to 64-bit applications. Cocoa provides archiving capabilities for both data and objects. For more information, see *Archives and Serializations Programming Guide for Cocoa*.

HIGeometry

Carbon: HIGeometry.h

The `HIGetScaleFactor` function returns the scale factor of an application's user interface. This function is not available to 64-bit applications, which must use Cocoa to implement their user interfaces. The Cocoa classes `NSScreen` and `NSWindow` have methods that return the same information.

HIObject

Carbon: HIObject.h

Some HIObject functions are not available to 64-bit applications. Refer to *HIObject Reference* for detailed information about function availability. In Cocoa, most objects are subclasses of the `NSObject` class. For more information about `NSObject`, see *Cocoa Fundamentals Guide*.

HIShape

ApplicationServices: HIShape.h

The `HIShapeSetQDClip` function sets the current clip in a graphics port to a shape. Since there is no QuickDraw graphics port in 64-bit applications, this function has been removed. You should use `HIShapeReplacePathInCGContext` followed by `CGContextClip` instead.

HIToolbar

Carbon: HIToolbar.h

HIToolbar is not available to 64-bit applications. Window toolbars are an integral feature in Cocoa. For more information, see *Toolbar Programming Topics for Cocoa*.

HIView

Carbon: HIView.h and related headers

HIView is not available to 64-bit applications. The `NSView` class in Cocoa defines the basic drawing, event-handling, and printing architecture of an application. For more information, see *Cocoa Fundamentals Guide* and *View Programming Guide for Cocoa*.

Interface Builder Services

Carbon: IBCarbonRuntime.h

Interface Builder Services includes functions such as `CreateNibReference` that are used to instantiate user interface objects from Carbon nib files.

Interface Builder Services is not available to 64-bit applications. If you're developing a 64-bit application, you should design a nib-based Cocoa user interface and use Cocoa to instantiate the user interface at runtime. To learn more about Cocoa nib files, see *Resource Programming Guide*.

Keyboard Layout Services

Carbon: Keyboards.h

All functions in Keyboard Layout Services except `KBGetLayoutType` have been deprecated and are not available to 64-bit applications. The replacement functions are in the new Text Input Source Services API. For more information, see *Text Input Source Services Reference*.

List Manager

Carbon: Lists.h

The List Manager is deprecated in Mac OS X v10.5 and is not available to 64-bit applications. The recommended replacements are the Cocoa classes `NSBrowser`, `NSTableView`, and `NSOutlineView`. For additional information about Cocoa browsers, see *Browsers*.

Menu Manager

Carbon: Menus.h

Most Menu Manager functions are not available to 64-bit applications. The contextual menu plug-in functionality is still available. Refer to *Menu Manager Reference* for detailed information about function availability. The Cocoa classes `NSMenu` and `NSMenuItem` are the basis for all types of menus in a Cocoa user interface. See *Application Menu and Pop-up List Programming Topics for Cocoa*.

Multilingual Text Engine (MLTE)

Carbon: MacTextEditor.h, HITextViews.h

MLTE is not available to 64-bit applications. You should use the Cocoa text system instead. For information about the Cocoa text system, see *Text System Overview*.

Navigation Services

Carbon: Navigation.h

Navigation Services is not available to 64-bit applications. Navigation Services dialogs are now implemented using the Cocoa classes `NSOpenPanel` and `NSSavePanel`. In 64-bit applications, you must use these classes directly.

Notification Manager

Carbon: Notification.h

The Notification Manager is not available to 64-bit applications. The recommended replacement is CFUserNotification, which is designed for use in applications that do not otherwise have user interfaces but may need occasional interaction with the user. For more information, see *CFUserNotification Reference*.

For information about notifications in Cocoa, see *Notification Programming Topics for Cocoa*. For general guidelines about user notifications from background processes and applications, see *Apple Human Interface Guidelines*.

Scrap Manager

Carbon: Scrap.h

The Scrap Manager is deprecated in Mac OS X v10.5 and is not available to 64-bit applications. The recommended replacements are the Pasteboard Manager or the Cocoa class `NSPasteboard`. For more information, see *Pasteboard Manager Programming Guide* or *Pasteboard Programming Topics for Cocoa*.

TextEdit Manager

Carbon: TextEdit.h, TSMTE.h

The TextEdit Manager and Text Services Manager for Text Edit (TSMTE) are not available to 64-bit applications. You should use Cocoa text views instead. For more information, see *Text Editing Programming Guide for Cocoa*.

Text Services Manager

Carbon: TextServices.h

Almost all Text Services Manager (TSM) functions are not available to 64-bit applications. The replacement for most of these functions is the new Text Input Source Services API. For more information, see *Text Input Source Services Reference*.

The `TSMSetInlineInputRegion` function, which depends on QuickDraw regions, is not available. A new Carbon event, `kEventTextInputIsMouseEventInInlineInputArea`, replaces this function.

The Text Services Manager Apple events are not sent or handled in 64-bit applications. These events were deprecated in Mac OS X v10.0. You should use the TSM Carbon events instead.

Window Manager

Carbon: MacWindows.h

The Window Manager is not available to 64-bit applications. For information about using windows in a Cocoa user interface, see *Window Programming Guide for Cocoa*.

Changes in Other C APIs

This chapter discusses 64-bit changes in other C APIs commonly used by Carbon applications. The APIs are listed in alphabetical order.

> **Note:** To find out if a specific function is not available to 64-bit applications, you can check the API reference documentation in the ADC Reference Library, use the Xcode 3.0 Research Assistant, or consult the relevant header file.

Alias Manager

CoreServices: Aliases.h

Alias Manager functions that use the `FSSpec` type or other deprecated types such as `CInfoPBPtr` are not available to 64-bit applications. Refer to *Alias Manager Reference* for detailed information about function availability.

Apple Event Manager

ApplicationServices: AppleEvents.h and related headers

Carbon: AEInteraction.h

If your 64-bit application uses the Apple Event Manager:

- For custom event types, you need to make sure that the data types they use have explicit sizes.

- You must use the preferred numeric Apple event descriptor types. For example, you need to use `typeSInt32` instead of `typeLongInteger`. For more information, see `ApplicationServices/AE/AEDataModel.h`.

- The use of the `FSSpec` type in Apple events is discouraged; you should change your code to use `FSRef`. Type coercions into `typeFSS` by Apple events are not supported.

- The amount of data passed in an Apple event is still restricted to 4 gigabytes.

Apple Graphics Library (AGL)

AGL: agl.h and related headers

Apple Graphics Library is a windowing system interface for OpenGL, specifically designed for the Carbon environment. AGL is not supported for use in 64-bit applications. For information about setting up an OpenGL drawing environment in Cocoa, see *Cocoa Drawing Guide*. For comprehensive information about OpenGL support in Mac OS X, and for detailed examples of how to integrate OpenGL into a Cocoa application, see *OpenGL Programming Guide for Mac OS X*.

Apple Type Services for Fonts

ApplicationServices: ATSFont.h, ATSTypes.h

Apple Type Services for Fonts functions that use the `FSSpec` type are deprecated and are not available to 64-bit applications. The `ATSPoint` data type is defined as `CGPoint` in 64-bit applications.

Apple Type Services for Unicode Imaging (ATSUI)

ApplicationServices: ATSUnicode.h and related headers

Most functions in ATSUI are not available to 64-bit applications. Refer to *ATSUI Reference* for detailed information about function availability. You should use Core Text or, if possible, the Cocoa text system. To learn about these technologies, see *Core Text Programming Guide* and *Text Layout Programming Guide for Cocoa*.

CFNetwork

CoreServices: CFNetwork.h and related headers

Several CFNetwork functions now return the standard `CFIndex` type. The `CFNetDiagnosticStatus` type has changed to use the `CFIndex` type.

Code Fragment Manager

CoreServices: CodeFragments.h

The Code Fragment Manager is not available to 64-bit applications. You should use the Mach-O executable format instead. For more information, see *Mach-O Programming Topics*.

Collection Manager

CoreServices: Collections.h

Unlike most other managers, the Collection Manager is not being expanded to 64-bit byte counts; it continues to take and return 32-bit byte counts. Function parameters that are pointers are, of course, expanded to 64 bits.

If you're currently using the Collection Manager, consider updating your software to use Core Foundation collection types such as CFMutableArray. For more information, see *Collections Programming Topics for Core Foundation*.

Color Picker Manager

Carbon: ColorPicker.h

Carbon applications use the Color Picker Manager to display a color selection window. The Color Picker Manager is not supported for use in 64-bit applications. You should use Cocoa to display the standard color selection window. For more information, see *Color Programming Topics for Cocoa*.

ColorSync Manager

ApplicationServices: ColorSync.h and related headers

Carbon: CMCalibrator.h

Many ColorSync Manager functions, data types, and constants that are rarely used, no longer recommended, or dependent on deprecated data types such as `FSSpec` are not available to 64-bit applications. For example, the following are no longer available:

■ Older versions of profile search and profile-identifier search functions. Use the profile iteration function `CMIterateColorSyncFolder` instead.

■ Older versions of functions that convert from one color model to another. Use the conversion functions in `CMFloatBitmap.h` instead.

■ Several functions that set default profiles.

■ The function `CMGetCWInfo`, because the data structure it returns is obsolete.

■ The entire color management scripting plug-in API. You should use Image I/O and Quartz 2D instead. For more information, see *Quartz 2D Programming Guide*.

■ The CMCalibrator API to make custom display color calibration plug-ins available in Displays Preferences. There is no replacement.

Component Manager

CoreServices: Components.h

Component Manager functions that use the `FSSpec` type, as well as four other obsolete functions, are not available to 64-bit applications. The following table lists these functions and their replacements:

Removed Function	Replacement Function
`RegisterComponentFile`	`RegisterComponentFileRef`
`RegisterComponentFileEntries`	`RegisterComponentFileRefEntries`
`ComponentFunctionImplemented`	`CallComponentCanDo`
`GetComponentVersion`	`CallComponentVersion`
`ComponentSetTarget`	`CallComponentTarget`
`GetComponentIconSuite`	`GetIconRefFromComponent`

On 64-bit architectures, the packing mechanism for the parameters described by the `ComponentParameters` structure has changed. The `params` array is a true array with 64-bit arguments. The number of arguments is computed by dividing the value in the `paramSize` field by `sizeof(long)`, which is 8 bytes in a 64-bit component. If you're writing a 64-bit component and you supply a glue function that's used to build the `ComponentParameters` structure, you need to update your glue code.

Core Audio

CoreAudio: CoreAudio.h and related headers

AudioToolbox: AudioToolbox.h and related headers

AudioUnit: AudioUnit.h and related headers

In the Core Audio framework, the Apple-supplied Carbon views (the generic Carbon view, and the EQ and DLS Synth custom views) are not available to 64-bit applications. In general, functions that use the `FSSpec` type are not available.

In the Audio Toolbox framework, the following are not available:

■ The API declared in `AUMIDIController.h`

■ The API declared in `DefaultAudioOutput.h`

■ Three functions in `MusicPlayer.h`:

❑ `MusicSequenceLoadSMF`

❑ `MusicSequenceLoadSMFData`

❑ `MusicSequenceSaveSMF`

■ Two functions in `AUGraph.h`:

❑ `AUGraphSetRenderNotification`

❑ `AUGraphRemoveRenderNotification`

In the Audio Unit framework, the following are not available:

- The API declared in `AUNTComponent.h` (Version 1 audio units created for Mac OS X v10.0 will not work in 64-bit applications)

- The function `AudioUnitRemovePropertyListener` in `AUComponent.h`

Core Foundation

CoreFoundation: CoreFoundation.h and related headers

Four Core Foundation scalar types are changing from 32 bits to 64 bits on 64-bit architectures: `CFTypeID`, `CFOptionFlags`, `CFHashCode`, and `CFIndex`. Although the `CFOptionFlags` type is growing, the high 32 bits may not be used for several years, because flag parameters defined in the upper range will not be usable by 32-bit programs.

Date, Time, and Measurement Utilities

CoreServices: DateTimeUtils.h, UTCUtils.h

Most of the functions in Date, Time, and Measurement Utilities have been superseded by Core Foundation functions and are not available to 64-bit applications. For information about replacements, see *Date, Time, and Measurement Utilities Reference*.

Desktop Manager

CoreServices: Files.h

The Desktop Manager is not available to 64-bit applications. You should use Icon Services and Launch Services instead. For more information, see *Obtaining and Using Icons With Icon Services* and *Launch Services Programming Guide*.

Device Manager

CoreServices: Files.h

The Device Manager is not available in Mac OS X v10.5 and later. The functions formerly declared in `Devices.h` have been moved into the 32-bit File Manager and deprecated. You should use the File Manager instead.

Dictionary Manager

ApplicationServices: Dictionary.h

The Dictionary Manager is deprecated in Mac OS X v10.5 and is not available to 64-bit applications. For related information, see "Language Analysis Manager" (page 34).

Dictionary Services lets you create your own custom dictionaries that users can access through the Dictionary application. You also use these services to access dictionaries programatically and to support user access to dictionary look-up through a contextual menu. To learn more about Dictionary Services, see *Dictionary Services Programming Guide*.

Display Manager

ApplicationServices: Displays.h

The Display Manager is deprecated and is not available to 64-bit applications. The `GDHandle` data type, and functions that use this type, are no longer available. You must use the `CGDirectDisplayID` data type and Quartz Display Services instead. For more information, see *Quartz Display Services Reference*.

File Manager

CoreServices: Files.h

In the File Manager, deprecated functions as well as a number of data types and constants are not available to 64-bit applications. Refer to *File Manager Reference* for detailed information about function availability.

The `FSSpec` data type is replaced by the opaque `FSRef` type. (The `FSSpec` type continues to exist so that functions that merely pass along an `FSSpec` object don't have to change.) Functions that actually use or provide `FSSpec` information are deprecated. Functions that operate on Pascal strings are also deprecated. The File Manager provides replacement functions that use the `FSRef` type and Unicode strings to facilitate the 64-bit transition.

Many functions that return reference numbers now return a standard data type called `FSIORefNum`, which is defined as a 16-bit value on 32-bit architectures and a 32-bit value on 64-bit architectures. Another new standard type, `FSVolumeRefNum`, is defined as a 16-bit value on all architectures. You should start using the `FSIORefNum` and `FSVolumeRefNum` types to store reference numbers in your applications.

Folder Manager

CoreServices: Folders.h

The deprecated functions in the Folder Manager are not available to 64-bit applications. Refer to *Folder Manager Reference* to learn which functions are deprecated.

A new function named `GetFolderNameUnicode` replaces `GetFolderName`.

Font Manager

ApplicationServices: Fonts.h

The Font Manager is not available to 64-bit applications. You should use Core Text instead. For more information, see *Core Text Programming Guide*.

Fonts Window Services

Carbon: FontPanel.h

Carbon applications use Fonts Window Services to display a font selection window. Fonts Window Services is not supported for use in 64-bit applications. You should use Cocoa to display the standard Fonts window. For more information, see *Font Panel*.

Icon Services and Utilities

ApplicationServices: Icons.h

CoreServices: IconsCore.h, IconStorage.h

Icon Utilities functions and data types, which are based on QuickDraw, are not available to 64-bit applications. These include data types that support classic icon formats such as `'ICON'`, `'cicn'`, and icon suites, and functions that use these data types. You must use the `IconRef` data type or a CGImage to represent an icon.

Deprecated functions in Icon Services are not available to 64-bit applications. Refer to *Icon Services and Utilities Reference* to learn which functions are deprecated.

Image Capture

Carbon: ImageCapture.h and related headers

In `ICAApplication.h`, the following opaque types are defined as `UInt32` instead of as pointers:

```
ICAConnectionID
ICAEventDataCookie
ICAObject
ICAProperty
ICAScannerSessionID
ICASessionID
```

Keychain Manager

Carbon: KeychainHI.h

CoreServices: KeychainCore.h

The Keychain Manager function `KCMakeKCRefFromFSSpec` is not available to 64-bit applications. Note that the Keychain Manager has been superseded by Keychain Services. For more information, see *Keychain Services Programming Guide*.

Language Analysis Manager

ApplicationServices: LanguageAnalysis.h

The Language Analysis Manager is not available to 64-bit applications. You should use CFStringTokenizer in the Core Foundation framework instead. For more information, see *CFStringTokenizer Reference*.

CFStringTokenizer was introduced in Mac OS X v10.5 to support typical tasks of internationalized applications. CFStringTokenizer let you tokenize a string of text into words, sentences or paragraphs in a language-neutral way (that is, you can tokenize a string without knowing the language.) It supports languages such as Japanese and Chinese that do not delimit words by spaces, and it can de-compound German compounds. You can obtain a Latin transcription for tokens. CFStringTokenizer also provides language identification.

Launch Services

CoreServices: LaunchServices.h and related headers

In Launch Services, two obsolete fields in the `LSItemInfoRecord` data structure are not available to 64-bit applications.

Mathematical and Logical Utilities

CoreServices: FixMath.h, Math64.h, fp.h, ToolUtils.h

Several existing functions in Mathematical and Logical Utilities have been changed to take arguments of type `SInt32` instead of type `long`, or to return a value of type `SInt32` instead of type `long`. For example, see the functions `Fix2Long` and `Long2Fix` in `CoreServices/CarbonCore/FixMath.h`.

Memory Management Utilities

CoreServices: OSUtils.h

The deprecated functions in Memory Management Utilities are not available to 64-bit applications. Refer to *Memory Management Utilities Reference* to learn which functions are deprecated.

The function `MakeDataExecutable` is not available. You should use the BSD function `mprotect` instead.

Memory Manager

CoreServices: MacMemory.h

The deprecated Memory Manager functions are not available to 64-bit applications. Refer to *Memory Manager Reference* to learn which functions are deprecated. Several obsolete data structures such as `Zone` are also not available.

The `BlockMove`, `BlockMoveData`, and `BlockZero` family of functions are not available; use the BSD functions `memmove` and `bzero` instead.

Pointer-based data types such as `Ptr` and `Handle` are now 64-bit types.

Multiprocessing Services

CoreServices: Multiprocessing.h, MultiprocessingInfo.h

Two Multiprocessing Services data types formerly defined as `UInt32` have changed:

```
typedef ItemCount TaskStorageIndex;
typedef LogicalAddress TaskStorageValue;
```

The function `MPDataToCode` is not available to 64-bit applications. You should use the BSD function `mprotect` instead.

Printing

In Mac OS X v10.5, the Carbon Printing Manager has been divided into two C APIs called Carbon Printing and Core Printing.

Carbon Printing is used by Carbon applications to present a user interface for printing. Carbon Printing includes functions to display the Page Setup and Print dialogs, and to execute a print loop that displays a print status dialog.

Core Printing is used for printing tasks that do not display a user interface. Any application can use this API, although Cocoa applications rarely need to use it.

Printing API changes that affect 64-bit applications are described below.

Carbon Printing

Carbon: PMApplication.h, PMApplicationDeprecated.h

Carbon Printing is not available to 64-bit applications. A 64-bit application with a Cocoa user interface should use Cocoa to display and customize printing dialogs, and to execute print jobs. For information on printing in Cocoa, see *Printing Programming Topics for Cocoa*.

Core Printing

ApplicationServices: PMCore.h, PMCoreDeprecated.h, PMDefinitions.h, PMDefinitionsDeprecated.h

Core Printing is available to 64-bit applications. However, most deprecated functions, data types, and constants are not available. Refer to *Core Printing Reference* for detailed information about function availability.

Deprecated functions include:

- Functions that use or assume the availability of QuickDraw

- Functions used to interleave PostScript and QuickDraw commands (the so-called "PICT with PostScript" document format)

- Functions that act on a current session rather than an explicit `PMPrintSession` object

- Functions that act on a current printer rather than an explicit `PMPrinter` object

- Functions that use or return Mac OS 9–style print records

- Functions that use or return handles

- A number of functions relevant only in Mac OS 9 and earlier

Other Printing Interfaces

Some additional changes have been made in the Mac OS X printing system that are related to 64-bit software development:

- You cannot build a 64-bit version of a printing dialog extension based on the CFPlugIn architecture described in *Extending Printing Dialogs*. If you're a printer vendor and you want to build a 64-bit version of a plug-in that adds custom panes to the Print dialog, you need to use the newer Cocoa-based printing dialog extension API. For more information, see the header file `Carbon/Print/PDEPluginInterface.h` and the sample project *OutputBinsPDE*, which is located in `/Developer/Examples/Printing/Printer`.

- To get information from PostScript printer description (PPD) files in a 64-bit application, you must use CUPS instead of PPDLib. See the functions, data types, and constants declared in `/usr/include/cups/ppd.h`.

- If you plan to write a 64-bit printer driver, you must write a CUPS filter instead of a traditional printer module. Note that there is no specific need to have a 64-bit driver unless you think you will get better performance or need access to larger amounts of memory.

- You cannot build a 64-bit version of a printer browser. You can use the function `PMServerLaunchPrinterBrowser` to display the standard printer browser.

Process Manager

ApplicationServices: Processes.h

Several Process Manager data structures with fields that refer to the `FSSpec` type are not available to 64-bit applications. The Process Manager provides 64-bit versions of these data structures with fields that refer to the `FSRef` type.

QuickDraw

ApplicationServices: QuickDraw.h and related headers

QuickDraw graphics ports are not supported. Consequently, the entire drawing API has been removed from 64-bit QuickDraw. Functions that manipulate regions and simple data structures (`Point` and `Rect`) are still available, but you cannot use QuickDraw to draw in a 64-bit application. To learn about other drawing APIs, see *Quartz Programming Guide for QuickDraw Developers* and *Cocoa Drawing Guide*.

PICT files cannot be rendered any more, not even with the function `QDPictDrawToCGContext`. They need to be converted to PDF or other image formats supported by Image I/O.

The Picture Utilities API is deprecated and is not available to 64-bit applications.

QuickTime

QuickTime: QuickTime.h and related headers

The QuickTime C APIs are not available to 64-bit applications. For these applications, the QuickTime Kit Objective-C classes are the primary interface into QuickTime. For example, if you use the QuickTime C API to play movies and you want to build a 64-bit version of your application, use the QuickTime Kit and display the movie in a Cocoa window. The QuickTime Kit is described in detail in *QuickTime Kit Programming Guide*.

Almost all of the QuickTime Kit classes, methods, and functions are supported, with one major exception: methods that take or return native QuickTime identifiers (in particular, `Movie`, `MovieController`, `Track`, and `Media`). For example, you cannot use the `quickTimeMovie` method in the `QTMovie` class to "dip down" into the QuickTime C APIs.

The following is a complete list of the QuickTime Kit methods that are not available to 64-bit applications.

In the `QTMovie` class:

```
+ (id)movieWithQuickTimeMovie:(Movie)movie
    disposeWhenDone:(BOOL)dispose
    error:(NSError **)errorPtr;
- (id)initWithQuickTimeMovie:(Movie)movie
    disposeWhenDone:(BOOL)dispose
    error:(NSError **)errorPtr;
- (Movie)quickTimeMovie;
- (MovieController)quickTimeMovieController;
```

In the `QTTrack` class:

```
+ (id)trackWithQuickTimeTrack:(Track)track
     error:(NSError **)errorPtr;
- (id)initWithQuickTimeTrack:(Track)track
     error:(NSError **)errorPtr;
- (Track)quickTimeTrack;
```

In the `QTMedia` class:

```
+ (id)mediaWithQuickTimeMedia:(Media)media
     error:(NSError **)errorPtr;
- (id)initWithQuickTimeMedia:(Media)media
     error:(NSError **)errorPtr;
- (Media)quickTimeMedia;
```

The QuickTime Music Architecture (QTMA) is not available to 64-bit applications. As an alternative, you should use Core Audio for audio and MIDI application development.

Resource Manager

CoreServices: Resources.h

Resource Manager functions that use the `FSSpec` type for creating and opening resource files are not available to 64-bit applications.

The Resource Manager introduces several new standard types:

- `ResFileRefNum` is an alias for `FSIORefNum` (see "File Manager" (page 32).) `ResFileRefNum` is used to return resource file reference numbers.

- `ResID` is used to represent resource IDs.

- `ResAttributes` is used to represent a resource attribute such as `resChangedBit`.

- `ResFileAttributes` is used to represent a resource file attribute such as `mapReadOnlyBit`.

- `ResCount` is used to represent a resource count.

- `ResourceIndex` is used to specify an item in a list of resources.

No code changes are necessary in order for your application to call functions that use these new types. You should adapt the `ResFileRefNum` type to declare local instances of resource reference numbers.

Script Manager

CoreServices: Script.h

The Script Manager is deprecated, and most Script Manager functions are not available to 64-bit applications. Refer to *Script Manager Reference* for detailed information about function availability.

Sound Manager

Carbon: Sound.h

The Sound Manager is deprecated in Mac OS X v10.5 and is not available to 64-bit applications. You should use Core Audio instead. For general information about Core Audio, see *Core Audio Overview*. For information about writing audio units, see *Audio Unit Programming Guide*.

Speech Recognition Manager

Carbon: SpeechRecognition.h

Some function parameter types have changed; for example, parameters of type `Size` are now of type `SInt32`. User-supplied callback data is now of type `SRefCon`.

Speech Synthesis Manager

ApplicationServices: SpeechSynthesis.h

A number of parameters and fields of type `long` are now of type `SInt32`. User-supplied callback data is now of type `SRefCon`.

Text Utilities

Carbon: TypeSelect.h

CoreServices: TextUtils.h, StringCompare.h, NumberFormatting.h

Most of the functions in Text Utilities are deprecated and are not available to 64-bit applications. (The only exception is the `Munger` utility function.) Refer to *Text Utilities Reference* to learn about replacements for deprecated functions.

Thread Manager

CoreServices: Threads.h

The underlying data type of `ThreadID` has changed from `UInt32` to `long`:

```
typedef unsigned long ThreadID;
```

As a result, the `ThreadID` type is still mapped directly to the `pthread_t` type in 64-bit applications.

Translation Manager

Carbon: Translation.h, TranslationExtensions.h

The Translation Manager and Translation Extensions are not available to 64-bit applications. You should use Translation Services instead. For more information, see the header file `ApplicationServices/HIServices/TranslationServices.h`.

vImage

Accelerate: vImage.h and related headers

The vImage framework declares the data type `vImage_AffineTransform` to serve the same purpose as the Quartz data type `CGAffineTransform`. The intent is that applications could use these types interchangeably. For example, you might use Quartz functions to operate on values of type `vImage_AffineTransform`.

In 64-bit applications, these data types are no longer interchangeable. Values of type `vImage_AffineTransform` contain single-precision quantities while values of type `CGAffineTransform` contain double-precision quantities. As a result, 64-bit applications that assume these types are interchangeable may fail.

Document Revision History

This table describes the changes to *64-Bit Guide for Carbon Developers*.

Date	Notes
2007-12-11	Added information about vImage.
2007-10-31	New document that explains the changes necessary to create a 64-bit executable version of a Carbon application.