ColorSync Manager Reference

Graphics & Imaging > ColorSync

2005-06-04

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

Contents

Tables

ColorSync Manager Reference

Framework:	ApplicationServices/ApplicationServices.h, Carbon/Carbon.h
Declared in	CMApplication.h CMCalibrator.h CMDeviceIntegration.h CMICCProfile.h CMMComponent.h CMScriptingPlugin.h CMTypes.h QuickdrawAPI.h

Overview

The ColorSync Manager is the API for ColorSync, a platform-independent color management system from Apple. ColorSync provides essential services for fast, consistent, and accurate color calibration, proofing, and reproduction using input, output, and display devices. ColorSync also provides an interface to system-wide color management settings that allows users to save color settings for specific jobs and switch between settings.

You need this reference if your software product performs color drawing, printing, or calculation, or if your peripheral device supports color. You also need this reference if you are creating a color management module (CMM)—a component that implements color-matching, color-conversion, and gamut-checking services.

The Color Picker Manager, documented separately, provides a standard user interface for soliciting color choices.

Carbon supports the majority of the ColorSync Manager programming interface. However, ColorSync 1.0 compatibility calls such as `CWNewColorWorld`, `GetProfile`, and `SetProfile` are not supported.

Nor does Carbon support ColorSync functions used for color management modules (CMMs). These functions aren't supported because Mac OS X uses Bundle Services to implement CMMs.

Some applications use the Component Manager to determine what CMMs are available. You cannot use the Component Manager for this purpose in Mac OS X. Apple has, however, provided a the function `CMIterateCMMInfo` to query for available CMMs.

Functions by Task

Accessing Profiles

`CMOpenProfile` (page 63)
Opens the specified profile and returns a reference to the profile.

`CMValidateProfile` (page 79)
Indicates whether the specified profile contains the minimum set of elements required by the current color management module (CMM) for color matching or color checking.

`CMCloseProfile` (page 26)
Decrements the reference count for the specified profile reference and, if the reference count reaches 0, frees all private memory and other resources associated with the profile.

`CMUpdateProfile` (page 78)
Saves modifications to the specified profile.

`CMCopyProfile` (page 28)
Duplicates the specified existing profile.

`CMProfileModified` (page 65)
Indicates whether the specified profile has been modified since it was created or last updated.

`CMGetProfileMD5` (page 48)
Gets the MD5 checksum from the profile header (message digest)

`CMGetProfileHeader` (page 47)
Obtains the profile header for the specified profile.

`CMSetProfileHeader` (page 76)
Sets the header for the specified profile.

`NCMGetProfileLocation` (page 88)
Obtains either a profile location structure for a specified profile or the size of the location structure for the profile.

`CMCloneProfileRef` (page 25)
Increments the reference count for the specified profile reference.

`CMGetProfileRefCount` (page 49)
Obtains the current reference count for the specified profile.

`CMFlattenProfile` (page 286) Deprecated in Mac OS X v10.5
Transfers a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document.

`CMGetProfileLocation` (page 293) Deprecated in Mac OS X v10.5
Obtains the location of a profile based on the specified profile reference.

`NCMUnflattenProfile` (page 318) Deprecated in Mac OS X v10.5
Unflattens a previouslyflattened profile.

Iterating Installed Profiles

`CMIterateColorSyncFolder` (page 57)
Iterates over the available profiles.

CMGetColorSyncFolderSpec (page 288) Deprecated in Mac OS X v10.5
Obtains the volume reference number and the directory ID for a ColorSync Profiles folder.

Creating Profiles

CMNewProfile (page 62)
Creates a new profile and associated backing copy.

NCWNewLinkProfile (page 92)
Obtains a profile reference for the specified by the profile location.

CMMakeProfile (page 60)
Makes a display or abstract profile by modifying an existing one.

CWNewLinkProfile (page 310) Deprecated in Mac OS X v10.5
Creates a device link profile based on the specified set of profiles.

Accessing Special Profiles

CMGetSystemProfile (page 54)
Obtains a reference to the current system profile.

CMGetDefaultProfileBySpace (page 33)
Gets the default profile for the specified color space.

CMGetDefaultProfileByUse (page 34)
Obtains the users' preferred device profile setting.

CMGetProfileByAVID (page 44)
Gets the current profile for a monitor.

CMSetProfileByAVID (page 72)
Sets the profile for the specified monitor, optionally setting video card gamma.

CMSetDefaultProfileBySpace (page 303) Deprecated in Mac OS X v10.5
Sets the default profile for the specified color space.

CMSetDefaultProfileByUse (page 304) Deprecated in Mac OS X v10.5
Sets values for device profile settings.

CMSetSystemProfile (page 306) Deprecated in Mac OS X v10.5
Sets the current system profile.

NCMSetSystemProfile (page 317) Deprecated in Mac OS X v10.5
Sets the location of a color profile.

Accessing Profile Elements

CMCountProfileElements (page 31)
Counts the number of elements in the specified profile.

CMProfileElementExists (page 65)
Tests whether the specified profile contains a specific element based on the element's tag signature.

CMGetProfileElement (page 46)
Obtains element data from the specified profile based on the specified element tag signature.

`CMSetProfileElement` (page 74)
Sets or replaces the element data for a specific tag in the specified profile.

`CMSetProfileElementSize` (page 76)
Reserves the element data size for a specific tag in the specified profile before setting the element data.

`CMGetPartialProfileElement` (page 44)
Obtains a portion of the element data from the specified profile based on the specified element tag signature.

`CMSetPartialProfileElement` (page 71)
Sets part of the element data for a specific tag in the specified profile.

`CMGetIndProfileElementInfo` (page 40)
Obtains the element tag and data size of an element by index from the specified profile.

`CMGetIndProfileElement` (page 38)
Obtains the element data corresponding to a particular index from the specified profile.

`CMSetProfileElementReference` (page 75)
Adds a tag to the specified profile to refer to data corresponding to a previously set element.

`CMRemoveProfileElement` (page 67)
Removes an element corresponding to a specific tag from the specified profile.

Accessing Profile Descriptions

`CMCopyProfileDescriptionString` (page 29)
Returns the name of a profile as a CFString.

`CMCopyProfileLocalizedString` (page 30)
Gets one specific string out of a profile

`CMCopyProfileLocalizedStringDictionary` (page 30)
Obtains a CFDictionary which contains the language locale and string for multiple localizations from a given tag.

`CMSetProfileLocalizedStringDictionary` (page 77)
Writes a dictionary of localized strings to a given tag in a profile.

`CMGetProfileDescriptions` (page 45)
Obtains the description tag data for a specified profile.

`CMSetProfileDescriptions` (page 73)
Sets the description tag data for a specified profile.

`CMGetScriptProfileDescription` (page 294) Deprecated in Mac OS X v10.5
Obtains the internal name (or description) of a profile and the script code identifying the language in which the profile name is specified from the specified profile.

Accessing Name-Class Profiles

`CMGetNamedColorInfo` (page 41)
Obtains information about a named color space from its profile reference.

`CMGetNamedColorValue` (page 43)
Obtains device and PCS color values for a specific color name from a named color space profile.

`CMGetIndNamedColorValue` (page 38)
Obtains device and PCS color values for a specific named color index from a named color space profile.

`CMGetNamedColorIndex` (page 40)
Obtains a named color index for a specific color name from a named color space profile.

`CMGetNamedColorName` (page 42)
Obtains a named color name for a specific named color index from a named color space profile.

Working With ColorWorlds

`NCWNewColorWorld` (page 90)
Creates a color world for color matching based on the specified source and destination profiles.

`CWConcatColorWorld` (page 83)
Sets up a color world that includes a set of profiles for various color transformations among devices in a sequence.

`NCWConcatColorWorld` (page 89)
Defines a color world for color transformations among a series of concatenated profiles.

`CWDisposeColorWorld` (page 84)
Releases the private storage associated with a color world when your application has finished using the color world.

`CWMatchColors` (page 87)
Matches colors in a color list, using the specified color world.

`CWCheckColors` (page 81)
Tests a list of colors using a specified color world to see if they fall within the gamut of a destination device.

`CWMatchBitmap` (page 86)
Matches the colors of a bitmap to the gamut of a destination device using the profiles specified by a color world.

`CWCheckBitmap` (page 80)
Tests the colors of the pixel data of a bitmap to determine whether the colors map to the gamut of the destination device.

`CWFillLookupTexture` (page 85)
Fills a 3-D lookup texture from a color world.

`CMGetCWInfo` (page 289) Deprecated in Mac OS X v10.5
Obtains information about the color management modules (CMMs) used for a specific color world.

Converting Colors

`CMConvertFixedXYZToXYZ` (page 273) Deprecated in Mac OS X v10.5
Converts colors specified in XYZ color space whose components are expressed as Fixed XYZ 32-bit signed values of type `CMFixedXYZColor` to equivalent colors expressed as XYZ 16-bit unsigned values of type `CMXYZColor`.

`CMConvertHLSToRGB` (page 274) Deprecated in Mac OS X v10.5
Converts colors specified in the HLS color space to equivalent colors defined in the RGB color space.

`CMConvertHSVToRGB` (page 275) Deprecated in Mac OS X v10.5
Converts colors specified in the HSV color space to equivalent colors defined in the RGB color space.

`CMConvertLabToXYZ` (page 276) Deprecated in Mac OS X v10.5
Converts colors specified in the L*a*b* color space to the XYZ color space.

`CMConvertLuvToXYZ` (page 276) Deprecated in Mac OS X v10.5
Converts colors specified in the L*u*v* color space to the XYZ color space.

`CMConvertRGBToGray` (page 277) Deprecated in Mac OS X v10.5
Converts colors specified in the RGB color space to equivalent colors defined in the Gray color space.

`CMConvertRGBToHLS` (page 278) Deprecated in Mac OS X v10.5
Converts colors specified in the RGB color space to equivalent colors defined in the HLS color space.

`CMConvertRGBToHSV` (page 279) Deprecated in Mac OS X v10.5
Converts colors specified in the RGB color space to equivalent colors defined in the HSV color space when the device types are the same.

`CMConvertXYZToFixedXYZ` (page 280) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space whose components are expressed as XYZ 16-bit unsigned values of type `CMXYZColor` to equivalent colors expressed as 32-bit signed values of type `CMFixedXYZColor`.

`CMConvertXYZToLab` (page 280) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space to the L*a*b* color space.

`CMConvertXYZToLuv` (page 281) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space to the L*u*v* color space.

`CMConvertXYZToXYZ` (page 282) Deprecated in Mac OS X v10.5
Converts a source color to a destination color using the specified chromatic adaptation method.

`CMConvertXYZToYxy` (page 283) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space to the Yxy color space.

`CMConvertYxyToXYZ` (page 283) Deprecated in Mac OS X v10.5
Converts colors specified in the Yxy color space to the XYZ color space.

Working With CMMs

`CMIterateCMMInfo` (page 55)
Iterates through the color management modules installed on the system.

`CMGetPreferredCMM` (page 292) Deprecated in Mac OS X v10.5
Identifies the preferred CMM specified by the ColorSync control panel.

Working With PostScript

`CMGetPS2ColorSpace` (page 53)
Obtains color space element data in text format usable as the parameter to the PostScript `setColorSpace` operator, which characterizes the color space of subsequent graphics data.

`CMGetPS2ColorRenderingIntent` (page 51)
Obtains the rendering intent element data in text format usable as the parameter to the PostScript `findRenderingIntent` operator, which specifies the color-matching option for subsequent graphics data.

`CMGetPS2ColorRendering` (page 50)
Obtains the color rendering dictionary (CRD) element data usable as the parameter to the PostScript `setColorRendering` operator, which specifies the PostScript color rendering dictionary to use for the following graphics data.

`CMGetPS2ColorRenderingVMSize` (page 52)
Determines the virtual memory size of the color rendering dictionary (CRD) for a printer profile before your application or driver obtains the CRD and sends it to the printer.

Working With QuickDraw

`CMEnableMatchingComment` (page 265) Deprecated in Mac OS X v10.4
Inserts a comment into the currently open picture to turn matching on or off.

`CMEndMatching` (page 265) Deprecated in Mac OS X v10.4
Concludes a QuickDraw-specific ColorSync matching session initiated by a previous call to the `NCMBeginMatching` function.

`CWCheckPixMap` (page 266) Deprecated in Mac OS X v10.4
Checks the colors of a pixel map using the profiles of a specified color world to determine whether the colors are in the gamut of the destination device.

`CWMatchPixMap` (page 268) Deprecated in Mac OS X v10.4
Matches a pixel map in place based on a specified color world.

`NCMBeginMatching` (page 269) Deprecated in Mac OS X v10.4
Sets up a QuickDraw-specific ColorSync matching session, using the specified source and destination profiles.

`NCMDrawMatchedPicture` (page 271) Deprecated in Mac OS X v10.4
Matches a picture's colors to a destination device's color gamut, as the picture is drawn, using the specified destination profile.

`NCMUseProfileComment` (page 272) Deprecated in Mac OS X v10.4
Automatically embeds a profile or a profile identifier into an open picture.

Registering Devices

`CMRegisterColorDevice` (page 66)
Registers a device with ColorSync.

`CMUnregisterColorDevice` (page 78)
Unregisters a device.

Accessing Default Devices

`CMGetDefaultDevice` (page 32)
Gets the default device.

`CMSetDefaultDevice` (page 67)
Sets the default device.

Accessing Devices Profiles

`CMGetDeviceFactoryProfiles` (page 35)
Retrieves the original profiles for a given device.

`CMSetDeviceFactoryProfiles` (page 68)
Establishes the profiles used by a given device.

`CMGetDeviceDefaultProfileID` (page 34)
Gets the default profile ID for a given device.

`CMSetDeviceDefaultProfileID` (page 68)
Sets the default profile ID for a given device.

`CMSetDeviceProfile` (page 69)
Change the profile used by a given device.

`CMGetDeviceProfile` (page 36)
Gets a profile used by a given device.

`CMGetDeviceProfiles` (page 290) Deprecated in Mac OS X v10.5
Gets the profiles used by a given device.

`CMSetDeviceProfiles` (page 305) Deprecated in Mac OS X v10.5
Changes the profiles used by a given device.

Accessing Device State and Information

`CMGetDeviceState` (page 37)
Gets the state of a device.

`CMSetDeviceState` (page 70)
Sets the state of a device.

`CMGetDeviceInfo` (page 35)
Gets information about a specified device.

Iterating Over Devices and Device Profiles

`CMIterateColorDevices` (page 56)
Iterates through the color devices available on the system, returning device information to a callback you supply.

`CMIterateDeviceProfiles` (page 58)
Iterates through the device profiles available on the system and returns information about profiles of the devices to a callback you supply.

Working With Image Files

`CMCountImageProfiles` (page 284) Deprecated in Mac OS X v10.5
Obtains a count of the number of embeded profiles for a given image.

`CMEmbedImage` (page 286) Deprecated in Mac OS X v10.5
Embeds an image with an ICC profile.

`CMGetImageSpace` (page 291) Deprecated in Mac OS X v10.5
Returns the signature of the data color space in which the color values of colors in an image are expressed.

`CMGetIndImageProfile` (page 291) Deprecated in Mac OS X v10.5
Obtains a specific embeded profile for a given image.

`CMLinkImage` (page 294) Deprecated in Mac OS X v10.5
Matches an image file with a device link profile.

`CMMatchImage` (page 295) Deprecated in Mac OS X v10.5
Color matches an image file.

`CMProofImage` (page 300) Deprecated in Mac OS X v10.5
Proofs an image.

`CMSetIndImageProfile` (page 306) Deprecated in Mac OS X v10.5
Sets a specific embeded profile for a given image.

`CMUnembedImage` (page 307) Deprecated in Mac OS X v10.5
Removes any ICC profiles embeded in an image.

`CMValidImage` (page 309) Deprecated in Mac OS X v10.5
Validates the specified image file.

Working With Video Card Lookup Tables

`CMGetGammaByAVID` (page 37)
Obtains the gamma value for the specified display device.

`CMSetGammaByAVID` (page 71)
Sets the gamma for the specified display device.

Miscellaneous

`CMGetColorSyncVersion` (page 32)
Gets ColorSync version information.

`CMLaunchControlPanel` (page 59)
Launches the ColorSync preferences pane.

`CMCalibrateDisplay` (page 25)
Calibrates a display.

Working With Universal Procedure Pointers

`DisposeCMBitmapCallBackUPP` (page 311) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a bitmap callback.

`DisposeCMConcatCallBackUPP` (page 311) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback.

`DisposeCMFlattenUPP` (page 312) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a data-flattening callback.

`DisposeCMMIterateUPP` (page 312) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function.

`DisposeCMProfileAccessUPP` (page 313) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a profile-access callback.

`DisposeCMProfileFilterUPP` (page 313) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a profile-filter callback.

`DisposeCMProfileIterateUPP` (page 313) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a profile-iteration callback.

`InvokeCMBitmapCallBackUPP` (page 314) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a bitmap callback.

`InvokeCMConcatCallBackUPP` (page 314) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a progress-monitoring callback.

`InvokeCMFlattenUPP` (page 315) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a data-flattening callback.

`InvokeCMMIterateUPP` (page 315) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a a progress-monitoring callback for the `CMIterateCMMInfo` function.

`InvokeCMProfileAccessUPP` (page 316) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a profile-access callback.

`InvokeCMProfileFilterUPP` (page 316) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a profile-filter callback.

`InvokeCMProfileIterateUPP` (page 316) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a profile-iteration callback.

`NewCMBitmapCallBackUPP` (page 318) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a bitmap callback.

`NewCMConcatCallBackUPP` (page 319) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a progress-monitoring callback.

`NewCMFlattenUPP` (page 319) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a data-flattening callback.

`NewCMMIterateUPP` (page 320) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function.

`NewCMProfileAccessUPP` (page 320) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a profile-access callback.

`NewCMProfileFilterUPP` (page 321) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a profile-filter callback.

`NewCMProfileIterateUPP` (page 321) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a profile-iteration callback.

Not Recommended

`CMCreateProfileIdentifier` (page 285) Deprecated in Mac OS X v10.5
Creates a profile identifier for a specified profile.

`CMDisposeProfileSearch` (page 285) Deprecated in Mac OS X v10.5
: Frees the private memory allocated for a profile search after your application has completed the search.

`CMNewProfileSearch` (page 296) Deprecated in Mac OS X v10.5
: Searches the ColorSync Profiles folder and returns a list of 2.x profiles that match the search specification.

`CMProfileIdentifierFolderSearch` (page 298) Deprecated in Mac OS X v10.5
: Searches the ColorSync Profiles folder and returns a list of profile references, one for each profile that matches the specified profile identifier.

`CMProfileIdentifierListSearch` (page 299) Deprecated in Mac OS X v10.5
: Searches a list of profile references and returns a list of all references that match a specified profile identifier.

`CMSearchGetIndProfile` (page 302) Deprecated in Mac OS X v10.5
: Opens the profile corresponding to a specific index into a specific search result list and obtains a reference to that profile.

`CMSearchGetIndProfileFileSpec` (page 302) Deprecated in Mac OS X v10.5
: Obtains the file specification for the profile at a specific index into a search result.

`CMUpdateProfileSearch` (page 308) Deprecated in Mac OS X v10.5
: Searches the ColorSync Profiles folder and updates an existing search result obtained originally from the `CMNewProfileSearch` function.

Functions

CMCalibrateDisplay

Calibrates a display.

```
OSErr CMCalibrateDisplay (
   CalibratorInfo *theInfo
);
```

Parameters

theInfo
: A pointer to a calibrator info data structure that contains the necessary data for calibrating a display.

Return Value

An `OSErr` value.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`CMCalibrator.h`

CMCloneProfileRef

Increments the reference count for the specified profile reference.

```
CMError CMCloneProfileRef (
   CMProfileRef prof
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile whose reference count is incremented.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The ColorSync Manager keeps an internal reference count for each profile reference returned from a call to the `CMOpenProfile`, `CMNewProfile`, or `CMCopyProfile` functions. Calling the `CMCloneProfileRef` function increments the count; calling the function `CMCloseProfile` (page 26) decrements it. The profile remains open as long as the reference count is greater than 0, indicating that at least one routine retains a reference to the profile. When the count reaches 0, the ColorSync Manager releases all private memory, files, or resources allocated in association with that profile.

When your application creates a copy of an entire profile with `CMCopyProfile`, the copy has its own reference count. The `CMCloseProfile` routine should be called for the copied profile, just as for the original. When the reference count reaches 0, private resources associated with the copied profile are freed.

When your application merely duplicates a profile reference, as it may do to pass a profile reference to a synchronous or an asynchronous task, it should call `CMCloneProfileRef` to increment the reference count. Both the called task and the caller should call `CMCloseProfile` when finished with the profile reference.

In your application, you must make sure that `CMCloseProfile` is called once for each time a profile reference is created or cloned. Otherwise, the memory and resources associated with the profile reference may not be properly freed, or an application may attempt to use a profile reference that is no longer valid.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CMApplication.h`

CMCloseProfile

Decrements the reference count for the specified profile reference and, if the reference count reaches 0, frees all private memory and other resources associated with the profile.

```
CMError CMCloseProfile (
   CMProfileRef prof
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) that identifies the profile that may need to be closed.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The ColorSync Manager keeps an internal reference count for each profile reference returned from a call to the `CMOpenProfile` (page 63), `CMNewProfile` (page 62), `CMCopyProfile` (page 28), or `CWNewLinkProfile` (page 310) functions. Calling the function `CMCloneProfileRef` (page 25) increments the count; calling the `CMCloseProfile` function decrements it. The profile remains open as long as the reference count is greater than 0, indicating there is at least one remaining reference to the profile. When the count reaches 0, the ColorSync Manager releases all private memory, files, or resources allocated in association with that profile.

When the ColorSync Manager releases all private memory and resources associated with a profile, any temporary changes your application made to the profile are not saved unless you first call the `CMUpdateProfile` function to update the profile.

When your application passes a copy of a profile reference to an independent task, whether synchronous or asynchronous, it should call the function `CMCloneProfileRef` (page 25) to increment the reference count. Both the called task and the caller should call `CMCloseProfile` when finished with the profile reference.

You call `CMCloneProfileRef` after copying a profile reference, but not after duplicating an entire profile (as with the `CMCopyProfile` function).

When your application passes a copy of a profile reference internally, it may not need to call `CMCloneProfileRef`, as long as the application calls `CMCloseProfile` once for the profile.

In your application, make sure that `CMCloseProfile` is called once for each time a profile reference is created or cloned. Otherwise, the private memory and resources associated with the profile reference may not be properly freed, or an application may attempt to use a profile reference that is no longer valid.

If you create a new profile by calling the `CMNewProfile` function, the profile is saved to disk when you call the `CMCloseProfile` function unless you specified `NULL` as the profile location when you created the profile.

To save changes to a profile before closing it, use the function `CMUpdateProfile` (page 78).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CMApplication.h`

CMCopyProfile

Duplicates the specified existing profile.

```
CMError CMCopyProfile (
   CMProfileRef *targetProf,
   const CMProfileLocation *targetLocation,
   CMProfileRef srcProf
);
```

Parameters

targetProf

A pointer to a profile reference of type `CMProfileRef` (page 166). On return, points to the profile copy that was created.

targetLocation

A pointer to a location specification that indicates the location, such as in memory or on disk, where the ColorSync Manager is to create the copy of the profile. A profile is commonly disk-file based. However, to accommodate special requirements, you can create a handle- or pointer-based profile, you can create a profile that is accessed through a procedure provided by your application, or you can create a temporary profile that is not saved after you call the `CMCloseProfile` function. To create a temporary profile, you either specify `cmNoProfileBase` as the kind of profile in the profile location structure or specify `NULL` for this parameter. To specify the location, you use the data type `CMProfileLocation` (page 165).

srcProf

A profile reference to the profile to duplicate.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMCopyProfile` function duplicates an entire open profile whose reference you specify. If you have made temporary changes to the profile, which you have not saved by calling `CMUpdateProfile`, those changes are included in the duplicated profile. They are not saved to the original profile unless you call `CMUpdateProfile` for that profile.

The ColorSync Manager maintains a modified flag to track whether a profile has been modified. After copying a profile, the `CMCopyProfile` function sets the value of the modified flag for that profile to `false`.

Unless you are copying a profile that you created, you should not infringe on copyright protection specified by the profile creator. To obtain the copyright information, you call the function `CMGetProfileElement` (page 46), specifying the `cmCopyrightTag` tag signature for the copyright element (defined in the `CMICCProfile.h` header file).

You should also check the `flags` field of the profile header structure `CM2Header` (page 116) for copyright information. You can test the `cmEmbeddedUseMask` bit of the flags field to determine whether the profile can be used independently. If the bit is set, you should use this profile as an embedded profile only and not copy the profile for your own purposes. The `cmEmbeddedUseMask` mask is described in "Flag Mask Definitions for Version 2.x Profiles" (page 224). The following code snippet shows how you might perform a test using the `cmEmbeddedUseMask` mask:

```
if (myCM2Header.flags & cmEmbeddedUseMask)
{
// profile should only be used as an embedded profile
}
else
```

```
{
// profile can be used independently
}
```

A calibration program, for example, might use the `CMCopyProfile` function to copy a device's original profile, then modify the copy to reflect the current state of the device. Or an application might want to copy a profile after unflattening it.

To copy a profile, you must obtain a reference to that profile by either opening the profile or creating it. To open a profile, use the function `CMOpenProfile` (page 63). To create a new profile, use the function `CMNewProfile` (page 62). As an alternative to using the `CMCopyProfile` function to duplicate an entire profile, you can use the same profile reference more than once. To do so, you call the function `CMCloneProfileRef` (page 25) to increment the reference count for the reference each time you reuse it. Calling the `CMCloneProfileRef` function increments the count; calling the function `CMCloseProfile` (page 26) decrements it. The profile remains open as long as the reference count is greater than 0, indicating at least one routine retains a reference to the profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMCopyProfileDescriptionString

Returns the name of a profile as a CFString.

```
CMError CMCopyProfileDescriptionString (
   CMProfileRef prof,
   CFStringRef *str
);
```

Parameters

prof

The profile to query.

str

On ouput, the name of the profile as a CFString.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
If the profile is localized, ColorSync obtains the best localized name for the current process.

Availability
Available in Mac OS X v. 10.3 and later.

Declared In
`CMApplication.h`

CMCopyProfileLocalizedString

Gets one specific string out of a profile

```
CMError CMCopyProfileLocalizedString (
   CMProfileRef prof,
   OSType tag,
   CFStringRef reqLocale,
   CFStringRef *locale,
   CFStringRef *str
);
```

Parameters

prof

The profile to query.

tag

The tag type of profile to query.

reqLocale

The requested locale (optional).

locale

On ouput, points to the locale (optional).

str

On output, points to the dictionary string (optional).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

For example, you pass in the optional tag 'dscm' plus "enUS" for the `reqLocale` parameter, to for a U.S. Enlish string. If a U.S. English string is not found, ColorSync falls back to a reasonable default:

```
err = CMCopyProfileLocalizedString (prof, 'dscm',
                CFSTR("enUS"), nil, &theStr);
```

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMApplication.h`

CMCopyProfileLocalizedStringDictionary

Obtains a CFDictionary which contains the language locale and string for multiple localizations from a given tag.

```
CMError CMCopyProfileLocalizedStringDictionary (
   CMProfileRef prof,
   OSType tag,
   CFDictionaryRef *theDict
);
```

Parameters

prof

The profile to query

tag

The tag type of profile to query

theDict

On output, points to the dictionary .See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

This function allows you to get a CFDictionary which contains the language locale and string for multiple localizations from a given tag.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMApplication.h`

CMCountProfileElements

Counts the number of elements in the specified profile.

```
CMError CMCountProfileElements (
   CMProfileRef prof,
   UInt32 *elementCount
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile to examine.

elementCount

A pointer to an element count. On return, a one-based count of the number of elements.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Every element in the profile outside the header is counted. A profile may contain tags that are references to other elements. These tags are included in the count.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetColorSyncVersion

Gets ColorSync version information.

```
CMError CMGetColorSyncVersion (
   UInt32 *version
);
```

Parameters

version

On output, points to the version of ColorSync installed on the system.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
`CMGetColorSyncVersion` relieves you from having to call `Gestalt` to find out the version of ColorSync installed on the system.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetDefaultDevice

Gets the default device.

```
CMError CMGetDefaultDevice (
   CMDeviceClass deviceClass,
   CMDeviceID *deviceID
);
```

Parameters

deviceClass

The device class whose default device you want to get. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

On return, points to the device ID for the default device.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
For each class of device, a device management layer may establish which of the registered devices is the default. This helps keep color management choices to a minimum and allows for some automatic features to be enabled, such as the "Default printer" as an output profile selection.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetDefaultProfileBySpace

Gets the default profile for the specified color space.

```
CMError CMGetDefaultProfileBySpace (
   OSType dataColorSpace,
   CMProfileRef *prof
);
```

Parameters

`dataColorSpace`

A four-character identifier of type `OSType`. You pass a color space signature that identifies the color space you wish to get the default profile for. The currently-supported values are `cmRGBData`, `cmCMYKData`, `cmLabData`, and `cmXYZData`. These constants are a subset of the constants described in "Color Space Signatures" (page 210). If you supply a value that is not supported, the `CMGetDefaultProfileBySpace` function returns an error value of `paramErr`.

`prof`

A pointer to a profile reference. On return, the reference specifies the current profile for the color space specified by `dataColorSpace`. `CMGetDefaultProfileBySpace` currently supports only file-based profiles.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMGetDefaultProfileBySpace` function currently supports the RGB, CMYK, Lab, and XYZ color spaces. The signature constants for these color spaces (shown above with the `dataColorSpace` parameter description) are described in "Color Space Signatures" (page 210). Support for additional color spaces may be provided in the future. `CMGetDefaultProfileBySpace` returns an error value of `paramErr` if you pass a color space constant it does not currently support.

The `CMGetDefaultProfileBySpace` function always attempts to return a file-based profile for a supported color space. For example, if the user has not specified a default profile in the ColorSync control panel for the specified color space, or if the profile is not found (the user may have deleted the profiles in the ColorSync Profiles folder or even the folder itself), `CMGetDefaultProfileBySpace` creates a profile, stores it on disk, and returns a reference to that profile. However, you should always check for an error return—for example, a user may have booted from a CD, so that `CMGetDefaultProfileBySpace` cannot save a profile file to disk.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetDefaultProfileByUse

Obtains the users' preferred device profile setting.

```
CMError CMGetDefaultProfileByUse (
   OSType use,
   CMProfileRef *prof
);
```

Parameters

use

A value that specifies the device type for which to obtain the profile.

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetDeviceDefaultProfileID

Gets the default profile ID for a given device.

```
CMError CMGetDeviceDefaultProfileID (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID *defaultProfID
);
```

Parameters

deviceClass

The device class to query. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID to query.

defaultID

On output, points to the id of the default profile for this device.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Device drivers and host software can set the default profile for a given device using the function `CMSetDeviceDefaultProfileID`.

Availability

Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetDeviceFactoryProfiles

Retrieves the original profiles for a given device.

```
CMError CMGetDeviceFactoryProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID *defaultProfID,
   UInt32 *arraySize,
   CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass

The device class to query. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID to query.

defaultProfID

A pointer to the default profile for this device.

arraySize

A pointer to the size of the array to be returned. You can first call this routine to get the size returned, then call it again with the size of the buffer to receive the array.

deviceProfiles

On output, points to the profile array. You can first pass `NULL` in this parameter to receive the size of the array in the `arraySize` parameter. Then, once the appropriate amount of storage has been allocated, a pointer to it can be passed in this parameter to have the array copied to that storage.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
This function allows you to retrieve the original profiles for a given device. These may differ from the actual profiles in use for that device, in the case where any factory profiles have been replaced (updated). To get the actual profiles in use, call `CMGetDeviceProfiles`.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetDeviceInfo

Gets information about a specified device.

```
CMError CMGetDeviceInfo (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceInfo *deviceInfo
);
```

Parameters

deviceClass

A device class to query. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

A device ID to query. You can pass `cmDefaultDeviceID`.

deviceInfo

On input, points to a device information dictionary On output, the dictionary is filled with device information. If, on input, `deviceInfo->deviceName` is `nil` then the name is not returned. If you wants the device name dictionary returned, you should provide in `deviceInfo->deviceName` the address where this routine should store the CFDictionaryRef. The caller is responsible for disposing of the name dictionary.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMGetDeviceProfile

Gets a profile used by a given device.

```
CMError CMGetDeviceProfile (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID profileID,
   CMProfileLocation *profileLoc
);
```

Parameters

deviceClass

The device class for the device whose profile you want to get. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID for the device whose profile you want to get.

defaultID

The ID of the default profile for this device.

deviceProfLoc

On return, the location of the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetDeviceState

Gets the state of a device.

```
CMError CMGetDeviceState (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceState *deviceState
);
```

Parameters

deviceClass

A device class to query. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

A device ID to query. You can pass `cmDefaultDeviceID`.

deviceState

On output, points to the device state. See "Device States" (page 221) for the values that can be returned.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetGammaByAVID

Obtains the gamma value for the specified display device.

```
CMError CMGetGammaByAVID (
   CMDisplayIDType theID,
   CMVideoCardGamma *gamma,
   UInt32 *size
);
```

Parameters

theID

A Display Manager ID value. You pass the ID value for the display device for which to set the gamma.

gamma

size

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetIndNamedColorValue

Obtains device and PCS color values for a specific named color index from a named color space profile.

```
CMError CMGetIndNamedColorValue (
   CMProfileRef prof,
   UInt32 index,
   CMColor *deviceColor,
   CMColor *PCSColor
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to a named color space profile to obtain color values from.

index

A one-based index value for a named color.

deviceColor

A pointer to a device color. On return, a device color value in `CMColor` union format. If the profile does not contain device values, `deviceColor` is undefined.

PCSColor

A pointer to a profile connection space color. On return, an interchange color value in `CMColor` union format.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Based on the passed named color index, the `CMGetIndNamedColorValue` function does a lookup into the named color tag and returns device and PCS values. If the index is greater than the number of named colors, `CMGetIndNamedColorValue` returns an error code.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetIndProfileElement

Obtains the element data corresponding to a particular index from the specified profile.

```
CMError CMGetIndProfileElement (
   CMProfileRef prof,
   UInt32 index,
   UInt32 *elementSize,
   void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile containing the element.

index

The index of the element whose data you want to obtain. This is a one-based element index within the range returned as the `elementCount` parameter of the `CMCountProfileElements` function.

elementSize

A pointer to an element data size. On input, specify the size of the element data to copy (except when `elementData` is set to `NULL`). Specify `NULL` to copy the entire element data. To obtain a portion of the element data, specify the number of bytes to be copy.

On return, the size of the element data actually copied.

elementData

A pointer to memory for element data. On input, you allocate memory. On return, this buffer holds the element data.

To obtain the element size in the `elementSize` parameter without copying the element data to this buffer, specify `NULL` for this parameter.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Before you call the `CMGetIndProfileElement` function to obtain the element data for an element at a specific index, you first determine the size in bytes of the element data. To determine the data size, you can

- call the function `CMGetIndProfileElementInfo` (page 40), passing the element's index
- call the `CMGetIndProfileElement` function itself, specifying a pointer to an unsigned long data type in the `elementSize` field and a `NULL` value in the `elementData` field

Once you have determined the size of the element data, you allocate a buffer to hold as much of the data as you need. If you want all of the element data, you specify `NULL` in the `elementSize` parameter. If you want only a portion of the element data, you specify the number of bytes you want in the `elementSize` parameter. You supply a pointer to the data buffer in the `elementData` parameter. After calling `CMGetIndProfileElement`, the `elementSize` parameter contains the size in bytes of the element data actually copied.

Before calling this function, you should call the function `CMCountProfileElements` (page 31). It returns the profile's total element count in the `elementCount` parameter.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetIndProfileElementInfo

Obtains the element tag and data size of an element by index from the specified profile.

```
CMError CMGetIndProfileElementInfo (
   CMProfileRef prof,
   UInt32 index,
   OSType *tag,
   UInt32 *elementSize,
   Boolean *refs
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile containing the element.

index

A one-based element index within the range returned by the `elementCount` parameter of the `CMCountProfileElements` function.

tag

A pointer to an element signature. On return, the tag signature of the element corresponding to the index.

elementSize

A pointer to an element size. On return, the size in bytes of the element data corresponding to the tag.

refs

A pointer to a reference count flag. On return, set to `true` if more than one tag in the profile refers to element data associated with the tag corresponding to the index.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The index order of elements is determined internally by the ColorSync Manager and is not publicly defined.

Before calling the `CMGetIndProfileElementInfo` function, you should call the function `CMCountProfileElements` (page 31), which returns the total number of elements in the profile in the `elementCount` parameter. The number you specify for the `index` parameter when calling `CMGetIndProfileElementInfo` should be in the range of 1 to `elementCount`; otherwise the function will return a result code of `cmIndexRangeErr`.

You might want to call this function, for example, to print out the contents of a profile.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetNamedColorIndex

Obtains a named color index for a specific color name from a named color space profile.

```
CMError CMGetNamedColorIndex (
   CMProfileRef prof,
   StringPtr name,
   UInt32 *index
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to a named color space profile to obtain a named color index from.

name

A pointer to a Pascal string. You supply a color name string value for the color to obtain the color index for.

index

A pointer to an index value. On return, an index value for a named color.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Based on the passed color name, the `CMGetNamedColorIndex` function does a lookup into the named color tag and, if the name is found in the tag, returns the index. Otherwise, `CMGetNamedColorIndex` returns an error code.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetNamedColorInfo

Obtains information about a named color space from its profile reference.

```
CMError CMGetNamedColorInfo (
   CMProfileRef prof,
   UInt32 *deviceChannels,
   OSType *deviceColorSpace,
   OSType *PCSColorSpace,
   UInt32 *count,
   StringPtr prefix,
   StringPtr suffix
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to a named color space profile to obtain named color information from.

deviceChannels

A pointer to a count value. On return, the number of device channels in the color space for the profile. It should agree with the "data color space" field in the profile header. For example, Pantone maps to CMYK, a four-channel color space. A value of 0 indicates no device channels were available.

deviceColorSpace

A pointer to a device color space. On return, a device color space, such as CMYK.

PCSColorSpace

A pointer to a profile connection space color space. On return, an interchange color space, such as Lab.

count

A pointer to a count value. On return, the number of named colors in the profile.

prefix

A pointer to a Pascal string. On return, the string contains a prefix, such as "Pantone", for each color name. The prefix identifies the named color system described by the profile.

suffix

A pointer to a Pascal string. On return, the string contains a suffix for each color name, such as "CVC".

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMGetNamedColorInfo` function returns information about the named color space referred to by the passed profile reference.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetNamedColorName

Obtains a named color name for a specific named color index from a named color space profile.

```
CMError CMGetNamedColorName (
   CMProfileRef prof,
   UInt32 index,
   StringPtr name
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to a named color space profile to obtain a named color name from.

index

An index value for a named color to obtain the color name for.

name

A pointer to a Pascal string. On return, a color name string.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Based on the passed color name index, the `CMGetNamedColorName` function does a lookup into the named color tag and returns the name. If the index is greater than the number of named colors, `CMGetNamedColorName` returns an error code.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetNamedColorValue

Obtains device and PCS color values for a specific color name from a named color space profile.

```
CMError CMGetNamedColorValue (
   CMProfileRef prof,
   StringPtr name,
   CMColor *deviceColor,
   CMColor *PCSColor
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 166) to a named color space profile to obtain color values from.

name
A pointer to a Pascal string. You supply a color name string for the color to get information for.

deviceColor
A pointer to a device color. On return, a device color value in `CMColor` union format. If the profile does not contain device values, `deviceColor` is undefined.

PCSColor
A pointer to a profile connection space color. On return, an interchange color value in `CMColor` union format.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Based on the passed color name, the `CMGetNamedColorValue` function does a lookup into the named color tag and, if the name is found in the tag, returns device and color PCS values. Otherwise, `CMGetNamedColorValue` returns an error code.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPartialProfileElement

Obtains a portion of the element data from the specified profile based on the specified element tag signature.

```
CMError CMGetPartialProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 offset,
   UInt32 *byteCount,
   void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile containing the target element.

tag

The tag signature for the element in question. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

offset

Beginning from the first byte of the element data, the offset from which to begin copying the element data.

byteCount

A pointer to a data byte count. On input, the number of bytes of element data to copy, beginning from the offset specified by the `offset` parameter. On return, the number of bytes actually copied.

elementData

A pointer to memory for element data. On input, you pass a pointer to allocated memory. On return, this buffer holds the element data.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMGetPartialProfileElement` function allows you to copy any portion of the element data beginning from any offset into the data. For the `CMGetPartialProfileElement` function to copy the element data and return it to you, your application must allocate a buffer in memory to hold the data.

You cannot use this function to obtain a portion of the `CM2Header` profile header. Instead, you must call the function `CMGetProfileHeader` (page 47) to get the entire profile header and read its contents.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileByAVID

Gets the current profile for a monitor.

```
CMError CMGetProfileByAVID (
   CMDisplayIDType theID,
   CMProfileRef *prof
);
```

Parameters

theAVID

A Display Manager ID value. You pass the ID value for the monitor for which to get the profile.

prof

A pointer to a profile reference. On return, a reference to the current profile for the monitor specified by `theAVID`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

If the Display Manager supports ColorSync, the `CMGetProfileByAVID` function calls on the Display Manager to get the profile for the specified display. This is the case if the version of the Display Manager is 2.2.5 or higher (if `gestaltDisplayMgrAttr` has the `gestaltDisplayMgrColorSyncAware` bit set).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileDescriptions

Obtains the description tag data for a specified profile.

```
CMError CMGetProfileDescriptions (
   CMProfileRef prof,
   char *aName,
   UInt32 *aCount,
   Str255 mName,
   ScriptCode *mCode,
   UniChar *uName,
   UniCharCount *uCount
);
```

Parameters

prof

A reference to the profile from which to obtain the description info.

aName

On output, a pointer to the profile name as a 7-bit Roman ASCII string.

aCount

On output, a pointer to a count of the number of characters returned in the `aName` field.

mName

On output, a pointer to the localized profile name string in Mac script-code format.

mCode

On output, a pointer the script code corresponding to the name string returned in the `mName` parameter.

uName

On output, a pointer to localizedUnicode profile name string.

uCount

On output, a pointer to a count of the number of Unicode (2-byte) characters returned in the `uName` parameter.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Use this function to get the description tag data for a given profile. The ICC Profile Format Specification (available at http://www.color.org) includes a description tag (`'desc'`), designed to provide more information about a profile than can be contained in a file name. This is especially critical on file systems with 8.3 names. The tag data can consist of up to three separate pieces (strings) of information for a profile. These different strings are designed to allow for display in different languages or on different computer systems. Applications typically use one of the strings to show profiles in a list or a pop-up menu.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileElement

Obtains element data from the specified profile based on the specified element tag signature.

```
CMError CMGetProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 *elementSize,
   void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile containing the target element.

tag

The tag signature (for example, 'A2B0', or constant `cmAToB0Tag`) for the element in question. The tag identifies the element. For a complete list of the public tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

elementSize

A pointer to a size value. On input, you specify the size of the element data to copy. Specify `NULL` to copy the entire element data. To obtain a portion of the element data, specify the number of bytes to copy.

On return, the size of the data returned.

elementData

A pointer to memory for element data. On input, you allocate memory. On return, this buffer holds the element data.

To obtain the element size in the `elementSize` parameter without copying the element data to this buffer, specify `NULL` for this parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Before you call the `CMGetProfileElement` function to obtain the element data for a specific element, you must know the size in bytes of the element data so you can allocate a buffer to hold the returned data.

The `CMGetProfileElement` function serves two purposes: to get an element's size and to obtain an element's data. In both instances, you provide a reference to the profile containing the element in the `prof` parameter and the tag signature of the element in the `tag` parameter.

To obtain the element data size, call the `CMGetProfileElement` function specifying a pointer to an unsigned long data type in the `elementSize` field and a `NULL` value in the `elementData` field.

After you obtain the element size, you should allocate a buffer large enough to hold the returned element data, then call the `CMGetProfileElement` function again, specifying `NULL` in the `elementSize` parameter to copy the entire element data and a pointer to the data buffer in the `elementData` parameter.

To copy only a portion of the element data beginning from the first byte, allocate a buffer the size of the number of bytes of element data you want to obtain and specify the number of bytes to copy in the `elementSize` parameter. In this case, On return the `elementSize` parameter contains the size in bytes of the element data actually returned.

You cannot use the `CMGetProfileElement` function to copy a portion of element data beginning from an offset into the data. To copy a portion of the element data beginning from any offset, use the function `CMGetPartialProfileElement` (page 44).

You cannot use this function to obtain a portion of the `CM2Header` profile header. Instead, you must call the function `CMGetProfileHeader` (page 47) to copy the entire profile header and read its contents.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetProfileHeader

Obtains the profile header for the specified profile.

```
CMError CMGetProfileHeader (
   CMProfileRef prof,
   CMAppleProfileHeader *header
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile whose header is to be copied.

header

A pointer to a profile header. On input, depending on the profile version, you may allocate a ColorSync 2.x or 1.0 header. On return, contains the profile data. For information about the ColorSync 2.x profile header structure, see `CM2Header` (page 116). For information about the ColorSync 1.0 header, see `CMHeader` (page 139).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMGetProfileHeader` function returns the header for a ColorSync 2.x or ColorSync 1.0 profile. To return the header, the function uses a union of type `CMAppleProfileHeader` (page 122), with variants for version 1.0 and 2.x headers.

A 32-bit version value is located at the same offset in either header. For ColorSync 2.x profiles, this is the `profileVersion` field. For ColorSync 1.0 profiles, this is the `applProfileVersion` field. You can inspect the value at this offset to determine the profile version, and interpret the remaining header fields accordingly.

To copy a profile header to a profile after you modify the header's contents, use the function `CMSetProfileHeader` (page 76).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileMD5

Gets the MD5 checksum from the profile header (message digest)

```
CMError CMGetProfileMD5 (
   CMProfileRef prof,
   CMProfileMD5 digest
);
```

Parameters

prof

digest

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
You can call this function to determine if two profiles are identical, or if a profile has changed over time. You can access this new MD5 checksum directly in the profile header, or use the function `CMGetProfileMD5`. This function has the advantage that it works with both ICC 4 profiles and earlier profiles.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMApplication.h`

CMGetProfileRefCount

Obtains the current reference count for the specified profile.

```
CMError CMGetProfileRefCount (
   CMProfileRef prof,
   long *count
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 166) to the profile whose reference count is obtained.

count
A pointer to a reference count. On return, the reference count for the specified profile reference.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The ColorSync Manager keeps an internal reference count for each profile reference returned from calls such as `CMOpenProfile` (page 63) or `CMNewProfile` (page 62). Calling the function `CMCloneProfileRef` (page 25) increments the count; calling the function `CMCloseProfile` (page 26) decrements it. The profile remains open as long as the reference count is greater than 0, indicating at least one routine retains a reference to the profile. When the count reaches 0, the ColorSync Manager releases all memory, files, or resources allocated in association with that profile.

An application that manages profiles closely can call the `CMGetProfileRefCount` function to obtain the reference count for a profile reference, then perform special handling if necessary, based on the reference count.

To copy a profile with the function `CMCopyProfile` (page 28), you must obtain a reference to that profile by either opening the profile or creating it. To open a profile, use the function `CMOpenProfile` (page 63). To create a new profile, use the function `CMNewProfile` (page 62). As an alternative to using the `CMCopyProfile` function to duplicate an entire profile, you can use the same profile reference more than once. To do so, you call the function `CMCloneProfileRef` (page 25) to increment the reference count for the reference each time you reuse it. Calling the `CMCloneProfileRef` function increments the count; calling the function `CMCloseProfile` (page 26) decrements it. The profile remains open as long as the reference count is greater than 0, indicating at least one routine retains a reference to the profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CMGetPS2ColorRendering

Obtains the color rendering dictionary (CRD) element data usable as the parameter to the PostScript setColorRendering operator, which specifies the PostScript color rendering dictionary to use for the following graphics data.

```
CMError CMGetPS2ColorRendering (
   CMProfileRef srcProf,
   CMProfileRef dstProf,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf

A profile reference to a profile that supplies the rendering intent for the CRD.

dstProf

A profile reference to a profile from which to extract the CRD data.

flags

If the value of flags is equal to cmPS8bit, the generated PostScript will utilize 8-bit encoding whenever possible to achieve higher data compaction. If the value of flags is not equal to cmPS8bit, the generated data will be 7-bit safe, in either ASCII or ASCII base-85 encoding.

proc

A pointer to a callback flatten function to perform the data transfer. For information, see the function CMFlattenProcPtr (page 96).

refCon

An untyped pointer to arbitrary data supplied by your application. CMGetPS2ColorSpace passes this data in calls to your CMFlattenProcPtr (page 96) function.

preferredCMMnotfound

A pointer to a flag for whether the preferred CMM was found. On return, has the value true if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value false.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The CMGetPS2ColorRendering function obtains CRD data from the profile specified by the dstProf parameter. To be valid, the parameter must specify an output profile with at most four components. The CMM uses the rendering intent from the profile specified by the srcProf parameter to determine which of the PostScript tags (ps2CR0Tag, ps2CR1Tag, ps2CR2Tag, or ps2CR3Tag) to use in creating the CRD. If none of these tags exists in the profile, the CMM creates the CRD from one of the multidimensional table tags (cmAToB0, cmAToB1, or cmAToB2), again chosen according to the rendering intent of the profile specified by the srcProf parameter.

This function is dispatched to the CMM component specified by the destination profile. If the designated CMM is not available or the CMM does not implement this function, the ColorSync Manager dispatches this function to the default CMM.

The CMM obtains the PostScript data and passes it to your low-level data transfer procedure, specified by the `proc` parameter. The CMM converts the data into a PostScript stream and calls your procedure as many times as necessary to transfer the data to it. Typically, the low-level data transfer function returns this data to the calling application or device driver to pass to a PostScript printer.

Before your application or device driver sends the CRD to the printer, it can call the function `CMGetPS2ColorRenderingVMSize` (page 52) to determine the virtual memory size of the CRD.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPS2ColorRenderingIntent

Obtains the rendering intent element data in text format usable as the parameter to the PostScript `findRenderingIntent` operator, which specifies the color-matching option for subsequent graphics data.

```
CMError CMGetPS2ColorRenderingIntent (
   CMProfileRef srcProf,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf
A profile reference to the source profile that defines the data color space and identifies the preferred CMM.

flags
If the value of `flags` is equal to `cmPS8bit`, the generated PostScript will utilize 8-bit encoding whenever possible to achieve higher data compaction. If the value of `flags` is not equal to `cmPS8bit`, the generated data will be 7-bit safe, in either ASCII or ASCII base-85 encoding.

proc
A low-level data transfer function supplied by the calling application to receive the PostScript data from the CMM. For more information, see the function `CMFlattenProcPtr` (page 96).

refCon
An untyped pointer to arbitrary data supplied by your application. `CMGetPS2ColorSpace` passes this data in calls to your `CMFlattenProcPtr` (page 96) function.

preferredCMMnotfound
A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMGetPS2ColorRenderingIntent` function obtains PostScript rendering intent information from the header of the source profile. It returns data by calling your low-level data transfer procedure and passing the PostScript data to it. Typically, your low-level data transfer function returns this data to the calling application or device driver to pass to a PostScript printer.

The `CMGetPS2ColorRenderingIntent` function is dispatched to the CMM component specified by the source profile. If the designated CMM is not available or the CMM does not implement this function, then ColorSync dispatches the function to the default CMM.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPS2ColorRenderingVMSize

Determines the virtual memory size of the color rendering dictionary (CRD) for a printer profile before your application or driver obtains the CRD and sends it to the printer.

```
CMError CMGetPS2ColorRenderingVMSize (
   CMProfileRef srcProf,
   CMProfileRef dstProf,
   UInt32 *vmSize,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf
A profile reference to a profile that supplies the rendering intent for the CRD.

dstProf
A profile reference to the destination printer profile.

vmSize
A pointer to a memory size. On return, the virtual memory size of the CRD.

preferredCMMnotfound
A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Your application or device driver can call this function to determine if the virtual memory size of the color rendering dictionary exceeds the printer's capacity before sending the CRD to the printer. If the printer's profile contains the Apple-defined optional tag `'psvm'` described in `CMConcatProfileSet` (page 128), then

the default CMM will return the data supplied by this tag specifying the CRD virtual memory size for the rendering intent's CRD. If the printer's profile does not contain this tag, then the CMM uses an algorithm to assess the VM size of the CRD, in which case the assessment can be larger than the actual maximum VM size.

The CMM uses the profile specified by the `srcProf` parameter to determine the rendering intent to use.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPS2ColorSpace

Obtains color space element data in text format usable as the parameter to the PostScript `setColorSpace` operator, which characterizes the color space of subsequent graphics data.

```
CMError CMGetPS2ColorSpace (
   CMProfileRef srcProf,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf

A profile reference to the source profile that defines the data color space and identifies the preferred CMM.

flags

If the value of `flags` is equal to `cmPS8bit`, the generated PostScript will utilize 8-bit encoding whenever possible to achieve higher data compaction. If the value of `flags` is not equal to `cmPS8bit`, the generated data will be 7-bit safe, in either ASCII or ASCII base-85 encoding.

proc

A pointer to a callback flatten function to receive the PostScript data from the CMM. For information, see the function `CMFlattenProcPtr` (page 96).

refCon

An untyped pointer to arbitrary data supplied by your application. `CMGetPS2ColorSpace` passes this data in calls to your `CMFlattenProcPtr` (page 96) function.

preferredCMMnotfound

A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMGetPS2ColorSpace` function obtains PostScript color space data from the source profile. The valid profile classes for the `CMGetPS2ColorSpace` function are display, input, and output profiles with at most four components.

To determine which profile elements to use to generate the PostScript color space data, the CMM:

- uses the PostScript `cmPS2CSATag`, if it exists
- otherwise, uses the multidimensional table tag (`cmAToB0`, `cmAToB1`, or `cmAToB2`), if it exists, for the rendering intent currently specified by the profile
- otherwise, uses the multidimensional table tag `cmAToB0`, if it exists
- otherwise, for display profiles only, uses the tristimulus tags (`cmRedColorantTag`, `cmGreenColorantTag`, `cmBlueColorantTag`) and the tonal curve tags (`cmRedTRCTag`, `cmGreenTRCTag`, and `cmBlueTRCTag`)

The CMM obtains the PostScript data from the profile and calls your low-level data transfer procedure passing the PostScript data to it. The CMM converts the data into a PostScript stream and calls your procedure as many times as necessary to transfer the data to it.

Typically, the low-level data transfer function returns this data to the calling application or device driver to pass to a PostScript printer as an operand to the PostScript `setcolorspace` operator, which defines the color space of graphics data to follow.

The `CMGetPS2ColorSpace` function is dispatched to the CMM component specified by the source profile. If the designated CMM is not available or the CMM does not implement this function, then the ColorSync Manager dispatches the function to the default CMM.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetSystemProfile

Obtains a reference to the current system profile.

```
CMError CMGetSystemProfile (
   CMProfileRef *prof
);
```

Parameters

prof

A pointer to a profile reference of type `CMProfileRef` (page 166). On return, a reference to the current system profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The following functions allow you to pass `NULL` as a parameter value to specify the system profile as a source or destination profile:

- `CMNewProfile` (page 62)
- `NCWNewColorWorld` (page 90)
- `NCMBeginMatching` (page 269)

- NCMDrawMatchedPicture (page 271)

Note that instead of passing NULL, you can pass a profile reference to a specific profile, including the system profile.

If you want to specify the system profile for any other function that requires a profile reference, such as CWConcatColorWorld (page 83) and CWNewLinkProfile (page 310), you must use an explicit reference. You can obtain such a reference with the CMGetSystemProfile function.

There are other reasons you might need to obtain a reference to the current system profile. For example, your application might need to display the name of the current system profile to a user.

To identify the location of the physical file, call the function CMGetProfileLocation (page 293).

When your application has finished using the current system profile, it must close the reference to the profile by calling the function CMCloseProfile (page 26).

Version Notes
Starting with version 2.5, use of the system profile has changed. So rather than call CMGetSystemProfile to obtain a reference to the system profile, you may be able to obtain a profile that is more appropriate for the current operation by calling CMGetDefaultProfileBySpace (page 33) to get the default profile for a color space or by calling CMGetProfileByAVID (page 44) to get the profile for a specific display.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CMIterateCMMInfo

Iterates through the color management modules installed on the system.

```
CMError CMIterateCMMInfo (
   CMMIterateUPP proc,
   UInt32 *count,
   void *refCon
);
```

Parameters

proc
A calling-program-supplied callback function that allows your application to monitor progress or abort the operation.

count
A pointer to the number of available CMMs.

refCon
A reference constant containing data specified by the calling application program.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMIterateCMMInfo` function returns information for all CMMs installed on the system. The caller can pass nil for the `CMMIterateUPP` param to simply get a count of CMMs. If a `CMMIterateUPP` proc is provided, it is called once for each CMM installed - with the `CMMInfo` structure filled accordingly. The caller can pass a data reference to `CMIterateCMMInfo` which will then be passed to the `CMMIterateUPP`. This might be used to allow some of the information in the `CMMInfo` data structure to be put into a menu, for example, by passing a menu reference as the refcon. Either the proc or the count parameter must be provided. The caller will get a `paramErr` if both are nil.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMIterateColorDevices

Iterates through the color devices available on the system, returning device information to a callback you supply.

```
CMError CMIterateColorDevices (
   CMIterateDeviceInfoProcPtr proc,
   UInt32 *seed,
   UInt32 *count,
   void *refCon
);
```

Parameters

proc
A pointer to a function that iterates through device information available on the system. This is optional, but allows you to obtain device information. If provided, your callback is invoked once for each registered device.

seed
A pointer to a seed value. This is optional. If you pass a pointer to a seed value that is the same as the current seed value, then the callback function specified by the `proc` parameter is not invoked.

count
On output, the number of color devices available on the system.

refCon
An optional value that passed to your callback.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
This routine gets device information about all registered color devices. If provided, the supplied callback functions is called once for each registered device, passing in the device info and the supplied refcon.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMIterateColorSyncFolder

Iterates over the available profiles.

```
CMError CMIterateColorSyncFolder (
   CMProfileIterateUPP proc,
   UInt32 *seed,
   UInt32 *count,
   void *refCon
);
```

Parameters

proc

A universal procedure pointer of type CMProfileIterateUPP, which is described in CMProfileIterateData (page 164). If you do not wish to receive callbacks, pass NULL for this parameter. Otherwise, pass a pointer to your callback routine.

seed

A pointer to a value of type long. The first time you call CMIterateColorSyncFolder, you typically set the value to 0. In subsequent calls, you set the value to the seed value obtained from the previous call. ColorSync uses the value in determining whether to call your callback routine, as described in the discussion for this function.

On return, the value is the current seed for the profile cache (unless you pass NULL, as described in the discussion).

count

A pointer to a value of type long. On return, the value is the number of available profiles. CMIterateColorSyncFolder provides the number of profiles even when no iteration occurs (unless you pass NULL, as described in the discussion below). To determine the count alone, without iteration, call CMIterateColorSyncFolder and pass a value of NULL for all parameters except count.

refCon

An untyped pointer to arbitrary data supplied by your application. CMIterateColorSyncFolder passes this data to your callback routine. If you pass NULL for the refCon parameter, CMIterateColorSyncFolder passes NULL to your callback routine.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Starting with ColorSync version 2.5, when your application needs information about the profiles currently available in the ColorSync Profiles folder, it can call the CMIterateColorSyncFolder routine, which in turn calls your callback routine once for each profile.

Even though there may be many profiles available, CMIterateColorSyncFolder can take advantage of ColorSync's profile cache to return profile information quickly, and (if the cache is valid) without having to open any profiles. For each profile, CMIterateColorSyncFolder supplies your routine with the profile header, script code, name, and location, in a structure of type CMProfileIterateData (page 164). As a result, your routine may be able to perform its function, such as building a list of profiles to display in a pop-up menu, without further effort (such as opening each file-based profile).

Only 2.x profiles are included in the profile search result.

Before calling `CMIterateColorSyncFolder` for the first time, you typically set `seed` to 0. ColorSync compares 0 to its current seed for the profile cache. It is not likely they will match—the odds are roughly one in two billion against it. If the values do not match, the routine iterates through all the profiles in the cache, calling your callback routine once for each profile. `CMIterateColorSyncFolder` then returns the actual seed value in `seed` (unless you passed `NULL` for that parameter).

If you pass the returned seed value in a subsequent call, and if there has been no change in the available profiles, the passed seed will match the stored cache seed and no iteration will take place.

Note that you can pass a `NULL` pointer for the `seed` parameter without harm. The result is the same as if you passed a pointer to 0, in that the function iterates through the available profiles, calling your callback routine once for each profile. However, the function does not return a seed value, since you have not passed a valid pointer.

You can force ColorSync to call your callback routine (if any profiles are available) by passing a `NULL` pointer or by passing 0 for the seed value. But suppose you have an operation, such as building a pop-up menu, that you only want to perform if the available profiles have changed. In that case, you pass the seed value from a previous call to `CMIterateColorSyncFolder`. If the profile folder has not changed, ColorSync will not call your callback routine.

Note that if there are no profiles available, ColorSync does not call your callback routine.

You can safely pass `NULL` for any or all of the parameters to the `CMIterateColorSyncFolder` function. If you pass `NULL` for all of the parameters, calling the function merely forces rebuilding of the profile cache, if necessary.

Version Notes
Starting with version 2.5, the name and location of the profile folder changed. In addition, the folder can now contain profiles within nested folders, as well as aliases to profiles or aliases to folders containing profiles. There are limits on the nesting of folders and aliases.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMIterateDeviceProfiles

Iterates through the device profiles available on the system and returns information about profiles of the devices to a callback you supply.

```
CMError CMIterateDeviceProfiles (
   CMIterateDeviceProfileProcPtr proc,
   UInt32 *seed,
   UInt32 *count,
   UInt32 flags,
   void *refCon
);
```

Parameters

proc

A pointer to a function that iterates through device information available on the system. This is optional, but allows you to obtain profile information for each device. If provided, your callback is invoked once for each registered device.

seed

A pointer to a seed value. This is optional. If you pass a pointer to a seed value that is the same as the current seed value, then the callback function specified by the `proc` parameter is not invoked.

count

On output, the number of color devices available on the system.

flags

A value that specifies which set of profiles you want to iterate through. It can have the following values: `cmIterateFactoryDeviceProfiles`, `cmIterateCustomDeviceProfiles`, `cmIterateCurrentDeviceProfiles`, `cmIterateAllDeviceProfiles` or `0`. Supplying `0` is the same as supplying `cmIterateCurrentDeviceProfiles`.

refCon

An optional value that passed to your callback.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMLaunchControlPanel

Launches the ColorSync preferences pane.

```
CMError CMLaunchControlPanel (
   UInt32 flags
);
```

Parameters

flags

A value that secifies how the preferences pane is launched. You currently must pass a vlaue of 0 for this parameter.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
When your application calls the function `CMLaunchControlPanel`, any changes made by the user will not be available (through calls such as `CMGetDefaultProfileBySpace`) until the user closes the ColorSync preferences pane. There is currently no ColorSync function that determines if the ColorSync preferences pane has been closed, but you can use the Process Manager API for this purpose.

Availability
Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMMakeProfile

Makes a display or abstract profile by modifying an existing one.

```
CMError CMMakeProfile (
   CMProfileRef prof,
   CFDictionaryRef spec
);
```

Parameters

prof
The profile to modify.

spec
A dictionary that specifies the modifications to make to the profile supplied in the `prof` parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The function `CMMakeProfile` adds appropriate tags to a profile to make a display or abstract profile based on a specification dictionary you supply.

One key in the specification dictionary must be `"profileType"` with a CFString value of either `"abstractLab"`, `"displayRGB"` or `"displayID"`.

The dictionary can optionally contain these keys-value pairs:

- `"description"`, with an associated CFString value
- `"copyright"`, with an associated CFString value

For a `profileType` key whose value is `"abstractLab"`, the dictionary can also contain the keys-value pairs listed in Table 1.

Table 1 Key-value pairs for "abstractLab"

Key	Value	Comment
`"gridPoints"`	A CFNumber (`SInt32`) that is an odd	Required
`"proc"`	A CFNumber (`SInt64`) coerced from a `LabToLabProcPtr` data type	Required

Key	Value	Comment
"refcon"	A CFNumber (SInt64) value coerced from a void* data type	Optional

For a profileType key whose value is "displayRGB", the dictionary can also contain the keys-value pairs listed in Table 2.

Table 2 Key-value pairs for "displayRGB"

Key	Value	Comment
"targetGamma"	A CFNumber (Float), for example, 1.8	Optional
"targetWhite"	A CFNumber (SInt32), for example, 6500	Optional
"gammaR	A CFNumber (Float), for example, 2.5	Required
"gammaG"	A CFNumber (Float), for example, 2.5	Required
"gammaB"	A CFNumber (Float), for example, 2.5	Required
"tableChans"	A CFNumber (SInt32), for example, 1 or 3	Optional
"tableEntries"	A CFNumber (SInt32), for example, 16 or 255	Optional
"tableEntrySize"	A CFNumber (SInt32), for example,1 or 2	Optional
"tableData"	A CFData (lut in RRRGGGBBB order)	Optional
"phosphorRx"	A CFNumber (Float)	Only if not supplying the phospherSet key.
phosphorRy"	A CFNumber (Float)	Only if not supplying the phospherSet key.
phosphorGx"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorGy"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorBx"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorBy"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorSet"	A CFString: "WideRGB", "700/525/450nm", "P22-EBU", "HDTV", "CCIR709", "sRGB", "AdobeRGB98" or "Trinitron"	Only if not supplying the phospher R, G, B keys
"whitePointx"	A CFNumber (Float)	Only if not supplying a whiteTemp key

Key	Value	Comment
"whitePointy"	A CFNumber (Float)	Only if not supplying a whiteTemp key
"whiteTemp"	A CFNumber (SInt32), for example, 5000, 6500, or 9300	Only if not supplying whitePointx and whitePointy keys

For a profileType key whose value is "displayID", the dictionary can also contain the keys-value pairs in Table 3

Table 3 Key-value pairs for "displayID"

Key	Value	Comment
"targetGamma"	A CFNumber (Float), for example, 1.8	Optional
"targetWhite"	A CFNumber (SInt32), for example, 6500	Optional
"displayID"	A CFNumber (SInt32)	Required

Optionally, the keys-value pairs s for a profileType key whose value is "displayRGB" can be provided to override the values from the display.

Availability
Available in Mac OS X v. 10.3 and later.

Declared In
CMApplication.h

CMNewProfile

Creates a new profile and associated backing copy.

```
CMError CMNewProfile (
   CMProfileRef *prof,
   const CMProfileLocation *theProfile
);
```

Parameters

prof

A pointer to a profile reference of type CMProfileRef (page 166). On return, a reference to the new profile.

theProfile

A pointer of type CMProfileLocation (page 165) to the profile location where the new profile should be created. A profile is commonly disk-file based—the disk file type for a profile is 'prof'. However, to accommodate special requirements, you can create a handle- or pointer-based profile, you can create a temporary profile that is not saved after you call the CMCloseProfile function, or you can create a profile that is accessed through a procedure provided by your application. To create a temporary profile, you either specify cmNoProfileBase as the kind of profile in the profile location structure or specify NULL for this parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMNewProfile` function creates a new profile and backing copy in the location you specify. After you create the profile, you must fill in the profile header fields and populate the profile with tags and their element data, and then call the function `CMUpdateProfile` (page 78) to save the element data to the profile file. The default ColorSync profile contents include a profile header of type `CM2Header` (page 116) and an element table.

To set profile elements outside the header, you use the function `CMSetProfileElement` (page 74), the function `CMSetProfileElementSize` (page 76), and the function `CMSetPartialProfileElement` (page 71). You set these elements individually, identifying them by their tag names.

When you create a new profile, all fields of the `CM2Header` profile header are set to 0 except the `size` and `profileVersion` fields. To set the header elements, you call the function `CMGetProfileHeader` (page 47) to get a copy of the header, assign values to the header fields, then call the function `CMSetProfileHeader` (page 76) to write the new header to the profile.

For each profile class, such as a device profile, there is a specific set of elements and associated tags, defined by the ICC, that a profile must contain to meet the baseline requirements. The ICC also defines optional tags that a particular CMM might use to optimize or improve its processing. You can also define private tags, whose tag signatures you register with the ICC, to provide a CMM with greater capability to refine its processing.

After you fill in the profile with tags and their element data, you must call the `CMUpdateProfile` function to write the new profile elements to the profile file.

This function is most commonly used by profile developers who create profiles for device manufacturers and by calibration applications. In most cases, application developers use existing profiles.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMOpenProfile

Opens the specified profile and returns a reference to the profile.

```
CMError CMOpenProfile (
   CMProfileRef *prof,
   const CMProfileLocation *theProfile
);
```

Parameters

prof

A pointer to a profile reference of type `CMProfileRef` (page 166). On return, the reference refers to the opened profile.

theProfile

A pointer to a profile location of type `CMProfileLocation` (page 165) for the profile to open. Commonly a profile is disk-file based, but it may instead be temporary, handle-based, pointer-based, or accessed through a procedure supplied by your application.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
If the `CMOpenProfile` function executes successfully, the profile reference refers to the opened profile. Your application uses this reference, for example, when it calls functions to color match, copy, and update a profile, and validate its contents.

The ColorSync Manager maintains private storage for each request to open a profile, allowing more than one application to use a profile concurrently.

When you create a new profile or modify the elements of an existing profile, the ColorSync Manager stores the new or modified elements in the private storage it maintains for your application. Any new or changed profile elements are not incorporated into the profile itself unless your application calls the function `CMUpdateProfile` (page 78) to update the profile. If you call the function `CMCopyProfile` (page 28) to create a copy of an existing profile under a new name, any changes you have made are incorporated in the profile duplicate but the original profile remains unchanged.

Before you call the `CMOpenProfile` function, you must set the `CMProfileLocation` data structure to identify the location of the profile to open. Most commonly, a profile is stored in a disk file. If the profile is in a disk file, use the profile location data type to provide its file specification. If the profile is in memory, use the profile location data type to specify a handle or pointer to the profile. If the profile is accessed through a procedure provided by your application, use the profile location data type to supply a universal procedure pointer to your procedure.

Your application must obtain a profile reference before you copy or validate a profile, and before you flatten the profile to embed it.

For example, your application can:

- open a profile
- call the `CMGetProfileHeader` function to obtain the profile's header to modify its values
- set new values
- call the `CMSetProfileHeader` function to replace the modified header
- pass the profile reference to a function such as `NCWNewColorWorld` (page 90) as the source or destination profile in a color world for a color-matching session
- When you close your reference to the profile by calling the function `CMCloseProfile` (page 26), your changes are discarded (unless you called the `CMUpdateProfile` function).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
CMApplication.h

CMProfileElementExists

Tests whether the specified profile contains a specific element based on the element's tag signature.

```
CMError CMProfileElementExists (
   CMProfileRef prof,
   OSType tag,
   Boolean *found
);
```

Parameters

prof

A profile reference of type CMProfileRef (page 166) that specifies the profile to examine.

tag

The tag signature (for example, 'A2B0', or constant cmAToB0Tag) for the element in question. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the CMICCProfile.h header file.

found

A pointer to a flag for whether the element was found. On return, the flag has the value true if the profile contains the element or false if it does not.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
You cannot use this function to test whether certain data in the CM2Header profile header exists. Instead, you must call the function CMGetProfileHeader (page 47) to copy the entire profile header and read its contents.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CMProfileModified

Indicates whether the specified profile has been modified since it was created or last updated.

```
CMError CMProfileModified (
   CMProfileRef prof,
   Boolean *modified
);
```

Parameters

prof

A profile reference of type CMProfileRef (page 166) to the profile to examine.

modified

A pointer to a Boolean variable. On return, the value of `modified` is set to `true` if the profile has been modified, `false` if it has not.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
When a profile is first opened, its modified flag is set to `false`. On calls that add to, delete from, or set the profile header or tags, the modified flag is set to `true`. After calling the function `CMUpdateProfile` (page 78), the modified flag is reset to `false`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMRegisterColorDevice

Registers a device with ColorSync.

```
CMError CMRegisterColorDevice (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CFDictionaryRef deviceName,
   const CMDeviceScope *deviceScope
);
```

Parameters

deviceSpec

The class of the device (e.g., 'scnr' ,'cmra' ,'prtr' ,'mntr').

deviceScope

The unique identifier of the class (Class + ID uniquely id's device).

deviceName

Name of the device. See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

deviceScope

Structure defining the user and host scope this device pertains to.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
For a device to be recognized by ColorSync (and possibly other parts of Mac OS X) it needs to register itself using this function. If the device has ColorSync profiles associated with it, it should identify those u after registering with this function. Once a device is registered, it can appear as an input, output, or proofing device in ColorSync controls, as long as it has profiles associated with it. Registration need only happen once, when the device is installed. Device drivers need not register their device each time they are loaded.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMRemoveProfileElement

Removes an element corresponding to a specific tag from the specified profile.

```
CMError CMRemoveProfileElement (
   CMProfileRef prof,
   OSType tag
);
```

Parameters

prof

A profile reference of type CMProfileRef (page 166) to the profile containing the tag remove.

tag

The tag signature for the element to remove.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The CMRemoveProfileElement function deletes the tag as well as the element data from the profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CMSetDefaultDevice

Sets the default device.

```
CMError CMSetDefaultDevice (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID
);
```

Parameters

deviceClass

The class of the device (e.g., 'scnr' ,'cmra' ,'prtr' ,'mntr').

deviceID

The unique identifier of the class (Class + ID uniquely id's device).

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
For each class of device, a device management layer may establish which of the registered devices is the default. This helps keep color management choices to a minimum and allows for some "automatic" features to be enabled, such as, "Default printer" as an output profile selection. If no such device (as specified by deviceClass and deviceID) has been registered, an error is returned.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMSetDeviceDefaultProfileID

Sets the default profile ID for a given device.

```
CMError CMSetDeviceDefaultProfileID (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID defaultProfID
);
```

Parameters

`deviceClass`
The device class for the device whose default profile you want to set. See "Device Classes" (page 220) for a list of the constants you can supply.

`deviceID`
The device ID for the device whose default profile you want to set.

`defaultID`
The ID of profile you want to set as the default.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The default profile ID for a given device is an important piece of information because of the function `CMGetProfileByUse`. The function `CMGetProfileByUse` returns the default profile for devices depending on the user's selection in the ColorSync preferences pane. Device drivers and host software can set the default profile for a given device using the function `CMSetDeviceDefaultProfileID`.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMSetDeviceFactoryProfiles

Establishes the profiles used by a given device.

```
CMError CMSetDeviceFactoryProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID defaultProfID,
   const CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass

The device class for the device whose factory profiles you want to establish. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID for the device whose factory profiles you want to establish.

defaultProfID

The ID of the default profile for this device.

deviceProfiles

On output, points to array that contains the factory device profiles.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

This function establishes the profiles used by a given device. It should be called after device registration to notify ColorSync of the device's profiles. Note that factory device profiles and the current device profiles might not be the same, since the latter may contain modifications to the factory set.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMSetDeviceProfile

Change the profile used by a given device.

```
CMError CMSetDeviceProfile (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   const CMDeviceProfileScope *profileScope,
   CMDeviceProfileID profileID,
   const CMProfileLocation *profileLoc
);
```

Parameters

deviceClass

The device class for the device whose profile you want to set. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID for the device whose profile you want to set.

profileScope

A pointer to the structure defining the scope this profile pertains to.

profileID

The ID of the default profile for this device.

deviceProfLoc

A pointer to the `CMProfileLocation` of the profile. Since this structure is a fixed length structure, you can simply pass a pointer to a stack-based structure or memory allocated for it.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

This function provides a way to change a profile used by a given device by ID. It can be called after device registration by calibration applications to reset a device's profile from factory defaults to calibrated profiles. In order for this call to be made successfully, you must pass the `CMDeviceClass` and `CMDeviceID` of the device being calibrated along with the `CMDeviceProfileID` of the profile to set. (Device selection and identification can be facilitated using the function `CMIterateColorDevices`). If an invalid `CMDeviceClass` or `CMDeviceID` is passed, an error (`CMInvalidDeviceClass` or `CMInvalidDeviceID`) is returned.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMSetDeviceState

Sets the state of a device.

```
CMError CMSetDeviceState (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceState deviceState
);
```

Parameters

deviceClass

The device class for the device whose state you want to set. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID for the device whose state you want to set.

deviceState

The device state to set. See "Device States" (page 221) for the values you can supply.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

This routines provides access for the device management layer to update the state of a particular device. For example, a device can be offline, busy, or calibrated. The state data passed in replaces the old state data with the value you supply.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMSetGammaByAVID

Sets the gamma for the specified display device.

```
CMError CMSetGammaByAVID (
   CMDisplayIDType theID,
   CMVideoCardGamma *gamma
);
```

Parameters

theID

A Display Manager ID value. You pass the ID value for the display device for which to set the gamma.

gamma

A pointer to the gamma value to which you want to set the display device.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetPartialProfileElement

Sets part of the element data for a specific tag in the specified profile.

```
CMError CMSetPartialProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 offset,
   UInt32 byteCount,
   const void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile containing the tag for which the element data is set.

tag

The tag signature for the element whose data is set. The tag identifies the element. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

offset

The offset in the existing element data where data transfer should begin.

byteCount

The number of bytes of element data to transfer.

elementData

A pointer to the buffer containing the element data to transfer to the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

You can use the `CMSetPartialProfileElement` function to set the data for an element when the amount of data is large and you need to copy it to the profile in segments.

After you set the element size, you can call this function repeatedly, as many times as necessary, each time appending a segment of data to the end of the data already copied until all the element data is copied.

If you know the size of the element data, you should call the function `CMSetProfileElementSize` (page 76) to reserve it before you call `CMSetPartialProfileElement` to set element data in segments. Setting the size first avoids the extensive overhead required to increase the size for the element data with each call to append another segment of data.

To copy the entire data for an element as a single operation, when the amount of data is small enough to allow this, call the function `CMSetProfileElement` (page 74).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileByAVID

Sets the profile for the specified monitor, optionally setting video card gamma.

```
CMError CMSetProfileByAVID (
   CMDisplayIDType theID,
   CMProfileRef prof
);
```

Parameters

theAVID

A Display Manager ID value. You pass the ID value for the monitor for which to set the profile.

prof

A profile reference. Before calling `CMSetProfileByAVID`, set the reference to identify the profile for the monitor specified by `theAVID`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
If you specify a profile that contains the optional profile tag for video card gamma, `CMSetProfileByAVID` extracts the tag and sets the video card based on the tag data. This is the only ColorSync function that sets video card gamma. The tag constant `cmVideoCardGammaTag` is described in "Video Card Gamma Tags" (page 260).

When a user sets a display profile using the Monitors & Sound control panel, the system profile is set to the same profile. When you call `CMSetProfileByAVID` to set a profile for a monitor, you may also wish to make that profile the system profile. If so, you must call `CMSetSystemProfile` (page 306) explicitly—calling `CMSetProfileByAVID` alone has no affect on the system profile.

Note that if the Display Manager supports ColorSync, the `CMSetProfileByAVID` function calls on the Display Manager to set the profile for the specified display. This is the case if the version of the Display Manager is 2.2.5 or higher (if `gestaltDisplayMgrAttr` has the `gestaltDisplayMgrColorSyncAware` bit set).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileDescriptions

Sets the description tag data for a specified profile.

```
CMError CMSetProfileDescriptions (
   CMProfileRef prof,
   const char *aName,
   UInt32 aCount,
   ConstStr255Param mName,
   ScriptCode mCode,
   const UniChar *uName,
   UniCharCount uCount
);
```

Parameters

prof
A reference to the profile into which to set the description tag data.

aName
A pointer to a 7-bit Roman ASCII profile name string to be set for the profile. This string must be null-terminated.

aCount
A count of the number of characters in the string specified in the `aName` parameter

mName
A pointer to the localized profile name string in Mac script-code format which is to be set for the profile. This string must be null-terminated.

mCode
The script code corresponding to the string specified by the `mName` parameter.

uName

A pointer to the localized Unicode profile name string which is to be set for the profile. This string must be null-terminated

uCount

A count of the number of Unicode characters in string specified by the `uName` parameter. Do not confuse this with a byte count, because each Unicode character requires two bytes.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Use this function to set the description tag data for a given profile. The ICC Profile Format Specification (available at http://www.color.org) includes a description tag (`'desc'`), designed to provide more information about a profile than can be contained in a file name. This is especially critical on file systems with 8.3 names. The tag data can consist of up to three separate pieces (strings) of information for a profile. These different strings are designed to allow for display in different languages or on different computer systems. Applications typically use one of the strings to show profiles in a list or a pop-up menu.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileElement

Sets or replaces the element data for a specific tag in the specified profile.

```
CMError CMSetProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 elementSize,
   const void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile containing the tag for which the element data is set.

tag

The tag signature for the element whose data is set. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

elementSize

The size in bytes of the element data set.

elementData

A pointer to the buffer containing the element data to transfer to the profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMSetProfileElement` function replaces existing element data if an element with the specified tag is already present in the profile. Otherwise, it sets the element data for a new tag. Your application is responsible for allocating memory for the buffer to hold the data to transfer.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileElementReference

Adds a tag to the specified profile to refer to data corresponding to a previously set element.

```
CMError CMSetProfileElementReference (
   CMProfileRef prof,
   OSType elementTag,
   OSType referenceTag
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 166) to the profile to add the tag to.

elementTag
The original element's signature tag corresponding to the element data to which the new tag will refer.

referenceTag
The new tag signature to add to the profile to refer to the element data corresponding to `elementTag`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
After the `CMSetProfileElementReference` function executes successfully, the specified profile will contain more than one tag corresponding to a single piece of data. All of these tags are of equal importance. Your application can set a reference to an element that was originally a reference itself without circularity.

If you call the function `CMSetProfileElement` (page 74) subsequently for one of the tags acting as a reference to another tag's data, then the element data you provide is set for the tag and the tag is no longer considered a reference. Instead, the tag corresponds to its own element data and not that of another tag.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileElementSize

Reserves the element data size for a specific tag in the specified profile before setting the element data.

```
CMError CMSetProfileElementSize (
   CMProfileRef prof,
   OSType tag,
   UInt32 elementSize
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile in which the element data size is reserved.

tag

The tag signature for the element whose size is reserved. The tag identifies the element. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

elementSize

The total size in bytes to reserve for the element data.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Your application can use the `CMSetProfileElementSize` function to reserve the size of element data for a specific tag before you call the function `CMGetPartialProfileElement` (page 44) to set the element data. The most efficient way to set a large amount of element data when you know the size of the data is to first set the size, then call the `CMSetPartialProfileElement` function to set each of the data segments. Calling the `CMSetProfileElementSize` function first eliminates the need for the ColorSync Manager to repeatedly increase the size for the data each time you call the `CMSetPartialProfileElement` function.

In addition to reserving the element data size, the `CMSetProfileElementSize` function sets the element tag, if it does not already exist.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileHeader

Sets the header for the specified profile.

```
CMError CMSetProfileHeader (
   CMProfileRef prof,
   const CMAppleProfileHeader *header
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile whose header is set.

header

A pointer to the new header to set for the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

You can use the `CMSetProfileHeader` function to set a header for a version 1.0 or a version 2.x profile. Before you call this function, you must set the values for the header, depending on the version of the profile. For a version 2.x profile, you use a data structure of type `CM2Header` (page 116). For a version 1.0 profile, you use a data structure of type `CMHeader` (page 139). You pass the header you supply in the `CMAppleProfileHeader` union, described in `CMAppleProfileHeader` (page 122).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileLocalizedStringDictionary

Writes a dictionary of localized strings to a given tag in a profile.

```
CMError CMSetProfileLocalizedStringDictionary (
   CMProfileRef prof,
   OSType tag,
   CFDictionaryRef theDict
);
```

Parameters

prof

The profile to modify.

tag

The tag type of profile to modify.

theDict

The dictionary to modify. See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In
CMApplication.h

CMUnregisterColorDevice

Unregisters a device.

```
CMError CMUnregisterColorDevice (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID
);
```

Parameters

deviceClass

The device class of the device you want to unregister. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID

The device ID of the device you want to unregister.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
When a device is no longer to be used on a system (as opposed to being offline), it should be unregistered. If a device is temporarily shut down or disconnected, it does not to be unregistered unless either of the following is true:

- The device driver is being removed (uninstalled)
- The device driver can't access the device profiles without the device

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMUpdateProfile

Saves modifications to the specified profile.

```
CMError CMUpdateProfile (
   CMProfileRef prof
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile to update.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMUpdateProfile` function makes permanent any changes or additions your application has made to the profile identified by the profile reference, if no other references to that profile exist.

The ColorSync Manager maintains a modified flag to track whether a profile has been modified. After updating a profile, the `CMUpdateProfile` function sets the value of the modified flag for that profile to `false`.

Each time an application calls the function `CMOpenProfile` (page 63), the function creates a unique reference to the profile. An application can also duplicate a profile reference by passing a copy to another task. You cannot use the `CMUpdateProfile` function to update a profile if more than one reference to the profile exists—attempting to do so will result in an error return. You can call the function `CMGetProfileRefCount` (page 49) to determine the reference count for a profile reference.

You cannot use the `CMUpdateProfile` function to update a ColorSync 1.0 profile.

After you fill in tags and their data elements for a new profile created by calling the function `CMNewProfile` (page 62), you must call the `CMUpdateProfile` function to write the element data to the new profile.

If you modify an open profile, you must call `CMUpdateProfile` to save the changes to the profile file before you call the function `CMCloseProfile` (page 26). Otherwise, the changes are discarded.

To modify a profile header, you use the function `CMGetProfileHeader` (page 47) and the function `CMSetProfileHeader` (page 76).

To set profile elements outside the header, you use the function `CMSetProfileElement` (page 74), the function `CMSetProfileElementSize` (page 76), and the function `CMSetPartialProfileElement` (page 71).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMValidateProfile

Indicates whether the specified profile contains the minimum set of elements required by the current color management module (CMM) for color matching or color checking.

```
CMError CMValidateProfile (
   CMProfileRef prof,
   Boolean *valid,
   Boolean *preferredCMMnotfound
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 166) to the profile to validate.

valid
A pointer to a valid profile flag. On return, has the value `true` if the profile contains the minimum set of elements to be valid and `false` if it does not.

preferredCMMnotfound

A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM specified by the profile was not available to perform validation or does not support this function and the default CMM was used. Has the value `false` if the profile's preferred CMM is able to perform validation.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
When your application calls the `CMValidateProfile` function, the ColorSync Manager dispatches the function to the CMM specified by the `CMMType` header field of the profile whose reference you specify. The preferred CMM can support this function or not.

If the preferred CMM supports this function, it determines if the profile contains the baseline elements for the profile class, which the CMM requires to perform color matching or gamut checking. For each profile class, such as a device profile, there is a specific set of required tagged elements defined by the ICC that the profile must include. The ICC also defines optional tags, which may be included in a profile. A CMM might use these optional elements to optimize or improve its processing. Additionally, a profile might include private tags defined to provide a CMM with processing capability particular to the needs of that CMM. The profile developer can define these private tags, register the tag signatures with the ICC, and include the tags in a profile. The CMM checks only for the existence of profile elements it does not check the element's content and size.

If the preferred CMM does not support the `CMValidateProfile` function request, the ColorSync Manager calls the default CMM to handle the validation request.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CWCheckBitmap

Tests the colors of the pixel data of a bitmap to determine whether the colors map to the gamut of the destination device.

```
CMError CWCheckBitmap (
   CMWorldRef cw,
   const CMBitmap *bitmap,
   CMBitmapCallBackUPP progressProc,
   void *refCon,
   CMBitmap *resultBitmap
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 183) to use for the color check.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

bitmap

A pointer to a bitmap of type `CMBitmap` (page 123) whose colors are to be checked.

progressProc

A calling program–supplied callback function that allows your application to monitor progress or abort the operation as the bitmap's colors are checked against the gamut of the destination device. The default CMM calls your function approximately every half-second unless color checking occurs in less time this happens when there is a small amount of data to be checked. If the function returns a result of `true`, the operation is aborted. Specify `NULL` for this parameter if your application will not monitor the bitmap color checking. For information on the callback function and its type definition, see the function `CMBitmapCallBackProcPtr` (page 93).

refCon

A pointer to a reference constant for application data passed as a parameter to calls to `progressProc`.

resultBitmap

A pointer to a bitmap. On return, contains the results of the color check. The bitmap must have bounds equal to the parameter of the source bitmap pointed to by `bitMap`. You must allocate the pixel buffer pointed to by the `image` field of the structure `CMBitmap` (page 123) and initialize the buffer to zeroes. Pixels are set to 1 if the corresponding pixel of the source bitmap indicated by `bitMap` is out of gamut. You must set the `space` field of the `CMBitMap` structure to `cmGamutResult1Space` color space storage format, as described in "Abstract Color Space Constants" (page 187).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

When your application calls the `CWCheckBitMap` function, the ColorSync Manager dispatches the function to the preferred CMM. The ColorSync Manager determines the preferred CMM based on the color world configuration. If the color world you pass in was created by the `CWConcatColorWorld` function, then the `keyIndex` field of the `CMConcatProfileSet` data structure identifies the preferred CMM. If the preferred CMM is not available, the default CMM is used to perform the color matching.

For the `CWCheckBitMap` function to execute successfully, the source profile's `dataColorSpace` field value and the `space` field value of the source bitmap pointed to by the `bitMap` parameter must specify the same data color space. `CWCheckBitMap` is not supported if the color world was initialized with a named color space profile.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CWCheckColors

Tests a list of colors using a specified color world to see if they fall within the gamut of a destination device.

```
CMError CWCheckColors (
   CMWorldRef cw,
   CMColor *myColors,
   size_t count,
   UInt8 *result
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 183) describing how the test is to occur.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

myColors

A pointer to an array containing a list of colors of type `CMColor` (page 125) to be checked . This function assumes the color values are specified in the data color space of the source profile.

count

The number of colors in the array. This is a one-based count.

result

A pointer to a buffer of packed bits. On return, each bit value is interpreted as a bit field with each bit representing a color in the array pointed to by `myColors`. You allocate enough memory to allow for 1 bit to represent each color in the `myColors` array. Bits in the `result` field are set to 1 if the corresponding color is out of gamut for the destination device. Ensure that the buffer you allocate is zeroed out before you call this function.

To access the packed bit-array, use code similar to the following:

```
inline bool GetNthBit (UInt8* result, int n)
{
    return ( 0 != (result[n/8] & (128>>(n%8))) );
}
```

The `result` bit array indicates whether the colors in the list are in or out of gamut for the destination profile. If a bit is set, its corresponding color falls out of gamut for the destination device. The leftmost bit in the field corresponds to the first color in the list.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The color test provides a preview of color matching using the specified color world.

All CMMs must support the `CWCheckColors` function.

If you have set a profile's gamut-checking mask so that no gamut information is included—see "Flag Mask Definitions for Version 2.x Profiles" (page 224) — `CWCheckColors` returns the `cmCantGamutCheckError` error.

The `CWCheckColors` function supports matching sessions set up with one of the multichannel color data types. `CWCheckColors` is not supported if the color world was initialized with a named color space profile.

Availability

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

Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CWConcatColorWorld

Sets up a color world that includes a set of profiles for various color transformations among devices in a sequence.

```
CMError CWConcatColorWorld (
   CMWorldRef *cw,
   CMConcatProfileSet *profileSet
);
```

Parameters

cw

A pointer to a color world. On return, a reference to a color world of type CMWorldRef (page 183). You pass the returned reference to other functions that use the color world for color-matching and color-checking sessions.

profileSet

A pointer of type CMConcatProfileSet (page 128) to an array of profiles describing the processing to carry out. You create the array and initialize it in processing order—source through destination.

You set the keyIndex field of the CMConcatProfileSet data structure to specify the zero-based index of the profile within the profile array whose specified CMM should be used for the entire color-matching or color-checking session. The profile header's CMMType field specifies the CMM. This CMM will fetch the profile elements necessary for the session.

Note that starting with ColorSync 2.5, the user can set a preferred CMM with the ColorSync control panel. If that CMM is available, ColorSync will use that CMM for all color conversion and matching operations the CMM is capable of performing.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The CWConcatColorWorld function sets up a session for color processing that includes a set of profiles. The array of profiles is in processing order—source through destination. Your application passes the function a pointer to a data structure of type CMConcatProfileSet to identify the profile array.

The quality flag setting—indicating normal mode, draft mode, or best mode—specified by the first profile prevails for the entire session the quality flags of following profiles in the sequence are ignored. The quality flag setting is stored in the flags field of the profile header. See CM2Header (page 116) and "Flag Mask Definitions for Version 2.x Profiles" (page 224) for more information on the use of flags.

The rendering intent specified by the first profile is used to color match to the second profile, the rendering intent for the second profile is used to color match to the third profile, and so on through the series of concatenated profiles.

The following rules govern the profiles you can specify in the profile array pointed to by the profileSet parameter for use with the CWConcatColorWorld function:

- In the profile array, you can pass in one or more profiles, but you must specify at least one profile. If you specify only one profile, it must be a device link profile. If you specify a device link profile, you cannot specify any other profiles in the profiles array; a device link profile must be used alone.

- In the profile array, you can specify an abstract profile anywhere in the sequence other than as the first or last profile.
- For the first and last profiles, you can specify device profiles or color space conversion profiles. However, when you set up a color-matching session with a named color space profile and other profiles, the named color profile must be first or the last profile in the color world—it cannot be in the middle.
- You cannot specify `NULL` to indicate the system profile. Note that starting with version 2.5, use of the system profile has changed.
- If you specify a color space profile in the middle of the profile sequence, it is ignored by the default CMM.
- If you specify a named color profile, it must be the first or the last profile. Otherwise, `CWConcatColorWorld` returns the value `cmCantConcatenateError`.

A after executing the `CWConcatColorWorld` function, you should call the function `CMCloseProfile` (page 26) for each profile to dispose of its reference.

Instead of passing in an array of profiles, you can specify a device link profile. For information on how to create a device link profile, see the `CWNewLinkProfile` function, which is described next.

Version Notes
The parameter description for `profileSet` includes changes in how this function is used starting with ColorSync version 2.5.

Note also that starting with version 2.5, use of the system profile has changed.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CWDisposeColorWorld

Releases the private storage associated with a color world when your application has finished using the color world.

```
void CWDisposeColorWorld (
   CMWorldRef cw
);
```

Parameters

cw

A color world reference of type `CMWorldRef` (page 183).

The function `NCWNewColorWorld` (page 90) and the function `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

Discussion
The following functions use color worlds. If you create a color world to pass to one of these functions, you must dispose of the color world when your application is finished with it.

- `CWMatchColors` (page 87)

- CWCheckColors (page 81)
- CWMatchBitmap (page 86)
- CWCheckBitmap (page 80)
- CWMatchPixMap (page 268)
- CWCheckPixMap (page 266)

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

Declared In
CMApplication.h

CWFillLookupTexture

Fills a 3-D lookup texture from a color world.

```
CMError CWFillLookupTexture (
   CMWorldRef cw,
   UInt32 gridPoints,
   UInt32 format,
   UInt32 dataSize,
   void *data
);
```

Parameters

cw
The color world to use.

gridPoints
The number of grid points per channel in the texture.

format
The format of pixels in texture; for example, cmTextureRGBtoRGBX8.

dataSize
The size in bytes of texture data to fill.

data
On output, points to the texture data to fill.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 261).

Discussion
You can use the resulting table in OpenGL to accelerate color management in hardware.

Availability
Available in Mac OS X v. 10.3 and later.

Declared In
CMApplication.h

CWMatchBitmap

Matches the colors of a bitmap to the gamut of a destination device using the profiles specified by a color world.

```
CMError CWMatchBitmap (
   CMWorldRef cw,
   CMBitmap *bitmap,
   CMBitmapCallBackUPP progressProc,
   void *refCon,
   CMBitmap *matchedBitmap
);
```

Parameters

cw

A reference to a color world of type `CMWorldRef` (page 183) in which matching is to occur.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

bitmap

A pointer to a bitmap of type `CMBitmap` (page 123) whose colors are to be matched.

progressProc

A calling program–supplied universal procedure pointer to a callback function that allows your application to monitor progress or abort the operation as the bitmap colors are matched. The default CMM calls your function approximately every half-second unless color matching occurs in less time this happens when there is a small amount of data to be matched. If the function returns a result of `true`, the operation is aborted. To match colors without monitoring the process, specify `NULL` for this parameter. For a description of the function your application supplies, see the function `CMBitmapCallBackProcPtr` (page 93).

refCon

A pointer to a reference constant for application data passed through as a parameter to calls to the `progressProc` function.

matchedBitmap

A pointer to a bitmap. On return, contains the color-matched image. You must allocate the pixel buffer pointed to by the `image` field of the structure `CMBitmap` (page 123). If you specify `NULL` for `matchedBitMap`, then the source bitmap is matched in place.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CWMatchBitmap` function matches a bitmap using the profiles specified by the given color world.

You should ensure that the buffer pointed to by the `image` field of the bitmap passed in the `bitMap` parameter is zeroed out before you call this function.

The ColorSync Manager does not explicitly support a CMY color space. However, for printers that have a CMY color space, you can use either of the following circumventions to make the adjustment:

- You can use a CMY profile, which the ColorSync Manager does support, with a CMYK color space. If you specify a CMYK color space in this case, the ColorSync Manager zeroes out the K channel to simulate a CMY color space.
- You can use an RGB color space and pass in the bitmap along with an RGB profile, then perform the conversion from RGB to CMY yourself.

For this function to execute successfully, the source profile's `dataColorSpace` field value and the `space` field value of the source bitmap pointed to by the `bitMap` parameter must specify the same data color space. Additionally, the destination profile's `dataColorSpace` field value and the `space` field value of the resulting bitmap pointed to by the `matchedBitMap` parameter must specify the same data color space, unless the destination profile is a named color space profile.

If you set `matchedBitMap` to `NULL` to specify in-place matching, you must be sure the space required by the destination bitmap is less than or equal to the size of the source bitmap.

Version Notes
The color spaces currently supported for the CWMatchBitmap function are defined in "Color Space Constants With Packing Formats" (page 203). Support for the following color space constants, was added with ColorSync version 2.5:

- `cmGray16Space`
- `cmGrayA32Space`
- `cmRGB48Space`.
- `cmCMYK64Space`
- `cmLAB48Space`

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CWMatchColors

Matches colors in a color list, using the specified color world.

```
CMError CWMatchColors (
   CMWorldRef cw,
   CMColor *myColors,
   size_t count
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 183) that describes how matching is to occur in the color-matching session.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

myColors

A pointer to an array containing a list of colors of type `CMColor` (page 125). On input, contains the list of colors to match. On return, contains the list of matched colors specified in the color data space of the color world's destination profile.

count

A one-based count of the number of colors in the color list of the `myColors` array.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CWMatchColors` function matches colors according to the profiles corresponding to the specified color world. On input, the color values in the `myColors` array are assumed to be specified in the data color space of the source profile. On return, the color values in the `myColors` array are transformed to the data color space of the destination profile.

All color management modules (CMM)s must support this function.

This function supports color-matching sessions set up with one of the multichannel color data types.

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

Declared In
`CMApplication.h`

NCMGetProfileLocation

Obtains either a profile location structure for a specified profile or the size of the location structure for the profile.

```
CMError NCMGetProfileLocation (
   CMProfileRef prof,
   CMProfileLocation *theProfile,
   UInt32 *locationSize
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166). Before calling `NCMGetProfileLocation`, you set the reference to specify the profile for which you wish to obtain the location or location structure size.

theProfile

A pointer to a profile location structure, as described in `CMProfileLocation` (page 165). If you pass `NULL`, `NCMGetProfileLocation` returns the size of the profile location structure for the profile specified by `prof` in the `locationSize` parameter. If you instead pass a pointer to memory you have allocated for the structure, on return, the structure specifies the location of the profile specified by `prof`.

locationSize

A pointer to a value of type `long`. If you pass `NULL` for the `profLoc` parameter, on return, `locationSize` contains the size in bytes of the profile location structure for the profile specified by `prof`. If you pass a pointer to a profile location structure in `profLoc`, set `locationSize` to the size of the structure before calling `NCMGetProfileLocation`, using the constant `cmCurrentProfileLocationSize`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `NCMGetProfileLocation` function is available starting with ColorSync version 2.5. It differs from its predecessor, `CMGetProfileLocation` (page 293), in that the newer version has a parameter for the size of the location structure for the specified profile.

You should use `NCMGetProfileLocation` rather than `CMGetProfileLocation` for the following reasons:

- Code using the older version (`CMGetProfileLocation`) may not be as easily ported to other platforms.
- Specifying the size of the profile location structure ensures that it can grow, if necessary, in the future.

The best way to use `NCMGetProfileLocation` is to call it twice:

1. Pass a reference to the profile to locate in the `prof` parameter and `NULL` for the `profLoc` parameter. `NCMGetProfileLocation` returns the size of the location structure in the `locationSize` parameter.
2. Allocate enough space for a structure of the returned size, then call the function again, passing a pointer in the `profLoc` parameter; on return, the structure specifies the location of the profile.

It is possible to call `NCMGetProfileLocation` just once, using the constant `cmCurrentProfileLocationSize` for the size of the allocated profile location structure and passing the same constant for the `locationSize` parameter. The constant `cmCurrentProfileLocationSize` may change in the future, but will be consistent within the set of headers you build your application with. However, if the size of the `CMProfileLocation` structure changes in a future version of ColorSync (and the value of `cmCurrentProfileLocationSize` as well) and you do not rebuild your application, `NCMGetProfileLocation` may return an error.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

NCWConcatColorWorld

Defines a color world for color transformations among a series of concatenated profiles.

```
CMError NCWConcatColorWorld (
   CMWorldRef *cw,
   NCMConcatProfileSet *profileSet,
   CMConcatCallBackUPP proc,
   void *refCon
);
```

Parameters

cw
A reference to a color world that the ColorSync Manager returns if the function completes successfully. You pass this reference to other functions that use the color world for color-matching and color-checking sessions.

profileSet
An array of profiles describing the processing to be carried out. The array is in processing order source through destination.

proc

A calling-program-supplied callback function that allows your application to monitor progress or abort the operation.

refCon

A reference constant containing data specified by the calling application program.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The caller can override the color management module (CMM) that would normally be selected by ColorSync by providing a CMM identifier in the `NCMConcatProfileSet` structure, or pass 0 to accept ColorSync's CMM selection (note that this could either be the user's preferred CMM selection or the CMM called for in the profile). The *flags* and k parameters are provided to allow easy customization of such attributes as quality and gamut-checking, while preserving the other settings. Each profile in the set can be customized by overriding the intent, and the selection of the transform tag. Together with other profiles, a custom-rendering environment can be set up to transform to or from device-dependent spaces with a minimum of gamut compression and/or unnecessary transformations to and from connection spaces. This flexibility comes at the price of specific knowledge of the profile contents and how device gamuts overlap.

Note that for standard input and output device profiles, A2B and B2A tags represent transforms from data space to connection space and from connection space to data space, respectively. Under these circumstances, the caller would not normally be able to use the same transform tags (e.g., `kUseAtoB`) consecutively, since a connection space would not be the same as the subsequent data space. If the spaces aren't the same, the caller will get `a cmCantConcatenateError` error returned. For profiles of type `cmLinkClass`, `cmAbstractClass`, `cmColorSpaceClass` , and `cmNamedColorClass` , these constants are not always meaningful, and the caller is encouraged to think in terms of the actual tags present in the profiles (e.g., A2B0 or B2A0). Under these conditions, it may well be appropriate to specify two transform tags of the same type consecutively, as long as the actual color spaces align in between tags. If this is not the case, a `cmCantConcatenateError` error is returned.

The callback proc is provided as protection against the appearance of a stalled machine during lengthy color world processing. If a CMM takes more than several seconds to process the information and create a color world, it will call the callback proc, if one is provided, and pass it the `refCon` provided. This is also true for `NCWNewLinkProfile`.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

NCWNewColorWorld

Creates a color world for color matching based on the specified source and destination profiles.

```
CMError NCWNewColorWorld (
   CMWorldRef *cw,
   CMProfileRef src,
   CMProfileRef dst
);
```

Parameters

cw

A pointer to a color world. On return, a reference to a matching session color world of type `CMWorldRef` (page 183). You pass this reference to other functions that use the color world.

src

A profile reference of type `CMProfileRef` (page 166) that specifies the source profile for the color-matching world. This profile's `dataColorSpace` element corresponds to the source data type for subsequent calls to functions that use this color world.

Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 33) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 44) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

dst

A profile reference of type `CMProfileRef` (page 166) that specifies the destination profile for the color-matching world. This profile's `dataColorSpace` element corresponds to the destination data type for subsequent calls to functions using this color world.

Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 33) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 44) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

You must set up a color world before your application can perform general purpose color-matching or color-checking operations. To set up a color world for these operations, your application can call `NCWNewColorWorld` after obtaining references to the profiles to use as the source and destination profiles for the color world. The following rules govern the types of profiles allowed:

- You can specify a device profile or a color space conversion profile for the source and destination profiles.
- You can not specify a device link profile or an abstract profile for either the source profile or the destination profile.
- Only one profile can be a named color profile.
- You can specify the system profile explicitly by reference or by giving `NULL` for either the source profile or the destination profile.

You should call the function `CMCloseProfile` (page 26) for both the source and destination profiles to dispose of their references after execution of the `NCWNewColorWorld` function.

The quality flag setting (indicating normal mode, draft mode, or best mode) specified by the source profile prevails for the entire session. The quality flag setting is stored in the `flags` field of the profile header. See `CM2Header` (page 116) and "Flag Mask Definitions for Version 2.x Profiles" (page 224) for more information on the use of flags. The rendering intent specified by the source profile also prevails for the entire session.

The function `CWConcatColorWorld` (page 83) also allocates a color world reference of type `CMWorldRef` (page 183).

Version Notes
The parameter descriptions for `src` and `dst` describe changes in how this functions is used starting with ColorSync version 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

NCWNewLinkProfile

Obtains a profile reference for the specified by the profile location.

```
CMError NCWNewLinkProfile (
   CMProfileRef *prof,
   const CMProfileLocation *targetLocation,
   NCMConcatProfileSet *profileSet,
   CMConcatCallBackUPP proc,
   void *refCon
);
```

Parameters

prof
The returned profile reference.

targetLocation
The location of the profile. Commonly a profile is disk-file based. However, the profile may be a file-based profile, a handle-based profile, or a pointer-based profile.

profileSet
A pointer to the profile set structure.

proc
A calling-program-supplied callback function that allows your application to monitor progress or abort the operation.

refCon
A reference constant containing data specified by the calling application program.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The same new flexibility in creating color worlds is extended to link profiles, which are not assumed to go from input device color space to output device color space. The returned profile is open, and should be closed when you are finished with it.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

Callbacks

CMBitmapCallBackProcPtr

Defines a pointer to a bitmap callback function that function reports on the progress of a color-matching or color-checking session being performed for a bitmap or a pixel map.

```
typedef Boolean (*MyCMBitmapCallBackProc)
(
    SInt16 progress,
    void * refCon
);
);
```

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

```
Boolean MyCMBitmapCallBackProc (
    SInt16 progress,
    void * refCon
);
```

Parameters

progress

A byte count that begins at an arbitrary value when the function is first called. On each subsequent call, the value is decremented by an amount that can vary from call to call, but that reflects how much of the matching process has completed since the previous call. If the function is called at all, it will be called a final time with a byte count of 0 when the matching is complete.

refCon

The pointer to a reference constant passed to your `MyCMBitmapCallBack` function each time the color management module (CMM) calls your function.

Return Value
`False` indicates the color-matching or color-checking session should continue. `True` indicates the session should be aborted—for example, the user may be holding down the Command–period keys.

Discussion
Your `MyCMBitmapCallBack` function allows your application to monitor the progress of a color-matching or color-checking session for a bitmap or a pixel map. Your function can also terminate the matching or checking operation.

Your callback function is called by the CMM performing the matching or checking process if your application passes a pointer to your callback function in the `progressProc` parameter when it calls one of the following functions: `CWCheckBitmap` (page 80) , `CWMatchBitmap` (page 86) , `CWCheckPixMap` (page 266) , and `CWMatchPixMap` (page 268). Note that your callback function may not be called at all if the operation completes in a very short period.

The CMM used for the color-matching session calls your function at regular intervals. For example, the default CMM calls your function approximately every half-second unless the color matching or checking occurs in less time; this happens when there is a small amount of data to match or check.

Each time the ColorSync Manager calls your function, it passes to the function any data stored in the reference constant. This is the data that your application specified in the `refCon` parameter when it called one of the color-matching or checking functions.

For large bitmaps and pixel maps, your application can display a progress bar or other indicator to show how much of the operation has been completed. You might, for example, use the reference constant to pass to the callback function a window reference to a dialog box. You obtain information on how much of the operation has completed from the `progress` parameter. The first time your callback is called, this parameter contains an arbitrary byte count. On each subsequent call, the value is decremented by an amount that can vary from call to call, but that reflects how much of the matching process has completed since the previous call. Using the current value and the original value, you can determine the percentage that has completed. If the callback function is called at all, it will be called a final time with a byte count of 0 when the matching is complete.

To terminate the matching or checking operation, your function should return a value of `true`. Because pixel-map matching is done in place, an application that allows the user to terminate the process should revert to the prematched image to avoid partial mapping.

For bitmap matching, if the `matchedBitMap` parameter of the `CWMatchBitmap` function specifies `NULL`, to indicate that the source bitmap is to be matched in place, and the application allows the user to abort the process, you should also revert to the prematched bitmap if the user terminates the operation.

Each time the ColorSync Manager calls your progress function, it passes a byte count in the `progress` parameter. The last time the ColorSync Manager calls your progress function, it passes a byte count of 0 to indicate the completion of the matching or checking process. You should use the 0 byte count as a signal to perform any cleanup operations your function requires, such as filling the progress bar to completion to indicate to the user the end of the checking or matching session, and then removing the dialog box used for the display.

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

Declared In
`CMTypes.h`

CMConcatCallBackProcPtr

Defines a pointer to a progress-monitoring function that the ColorSync Manager calls during lengthy color world processing.

```
typedef Boolean (*CMConcatCallBackProcPtr)
(
    SInt32 progress,
    void *refCon
);
```

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

```
Boolean MyCMConcatCallBackProc (
    SInt32 progress,
    void *refCon
);
```

Parameters

progress

refCon

Discussion
If a CMM takes more than several seconds to process the information and create a color world, it will call the Callback proc, if one is provided, and pass it the refCon provided

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

Declared In
`CMTypes.h`

CMCountImageProfilesProcPtr

Defines a pointer to a function that obtains a count of the number of embeded profiles for a given image..

```
typedef CMError (*CMCountImageProfilesProcPtr)
(
    const FSSpec * spec,
    UInt32 * count
);
```

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

```
CMError MyCMCountImageProfilesProc (
    const FSSpec * spec,
    UInt32 * count
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

count

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMEmbedImageProcPtr

Defines a pointer to a function that embeds an image with an ICC profile..

```
typedef CMError (*CMEmbedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    CMProfileRef embProf
);
```

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

```
CMError MyCMEmbedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    CMProfileRef embProf
);
```

Parameters

specFrom
See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

repl

embProf

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMFlattenProcPtr

Defines a pointer to a data transfer callback function that transfers profile data from the format for embedded profiles to disk file format or vice versa.

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

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

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

Parameters

command

The command with which the `MyCMFlattenCallback` function is called. This command specifies the operation the function is to perform.

size

A pointer to a size value. On input, the size in bytes of the data to transfer. On return, the size of the data actually transferred.

data

A pointer to the buffer supplied by the ColorSync Manager to use for the data transfer.

refCon

A pointer to a reference constant that holds the application data passed in from the functions `CMFlattenProfile` (page 286), `NCMUnflattenProfile` (page 318), `CMGetPS2ColorRenderingVMSize` (page 52), `CMGetPS2ColorRenderingIntent` (page 51), or `CMFlattenProfile` (page 286). Each time the CMM calls your `MyCMFlattenCallback` function, it passes this data to the function.

Starting in ColorSync version 2.5, the ColorSync Manager calls your function directly, without going through the preferred, or any, CMM.

Return Value

A result code. See "ColorSync Manager Result Codes" (page 261).

Discussion

IThis callback can be used, for example, by PostScript functions to transfer data from a profile to text format usable by a PostScript driver. Starting in ColorSync version 2.5, the ColorSync Manager calls your data transfer function directly, without going through the preferred, or any, CMM. So any references to the CMM in the discussion that follows are applicable only to versions of ColorSync prior to version 2.5. Where the discussion does not involve CMMs, it is applicable to all versions of ColorSync.

Your `MyCMFlattenCallback` function is called to flatten and unflatten profiles or to transfer PostScript-related data from a profile to the PostScript format to send to an application or device driver.

The ColorSync Manager and the CMM communicate with the `MyCMFlattenCallback` function using the command parameter to identify the operation to perform. To read and write profile data, your function must support the following commands: `cmOpenReadSpool`, `cmOpenWriteSpool`, `cmReadSpool`, `cmWriteSpool`, and `cmCloseSpool`.

You determine the behavior of your `MyCMFlattenCallback` function. The following sections describe how your function might handle the flattening and unflattening processes.

Flattening a Profile:

The ColorSync Manager calls the specified profile's preferred CMM when an application calls the `CMFlattenProfile` function to transfer profile data embedded in a graphics document.

The ColorSync Manager determines if the CMM supports the `CMFlattenProfile` function. If so, the ColorSync Manager dispatches the `CMFlattenProfile` function to the CMM. If not, ColorSync calls the default CMM, dispatching the `CMFlattenProfile` function to it.

The CMM communicates with the `MyCMFlattenCallback` function using a command parameter to identify the operation to perform. The CMM calls your function as often as necessary, passing to it on each call any data transferred to the CMM from the `CMFlattenProfile` function's `refCon` parameter.

The ColorSync Manager calls your function with the following sequence of commands: `cmOpenWriteSpool`, `cmWriteSpool`, and `cmCloseSpool`. Here is how you should handle these commands:

- When the CMM calls your function with the `cmOpenWriteSpool` command, you should perform any initialization required to write profile data you receive from the CMM to a buffer or file.
- The CMM will call your function with the `cmWriteSpool` command as many times as necessary to transfer all the profile data to you. Each time you are called, you should receive the data and write it to your buffer or file, returning in the `size` parameter the number of bytes of data you actually accepted.
- When the CMM calls your function with the `cmCloseSpool` command, you should perform any required cleanup processes.

As part of this process, your function can embed the profile data in a graphics document, for example, a PICT file or a TIFF file. For example, your `MyCMFlattenCallback` function can call the QuickDraw `PicComment` function to embed the flattened profile in a picture.

Unflattening a Profile:

When an application calls the `CMUnflattenProfile` function to transfer a profile that is embedded in a graphics document to an independent disk file, the ColorSync Manager calls your `MyCMFlattenCallback` function with the following sequence of commands: `cmOpenReadSpool`, `cmReadSpool`, `cmCloseSpool`. Here is how you should handle these commands:

- When the ColorSync Manager calls your function with the `cmOpenReadSpool` command, you should perform any initialization required to read from the embedded profile format.
- The ColorSync Manager calls your function with the `cmReadSpool` command as many times as necessary, directing your function to extract the profile data from the embedded format in the image file and return it to the ColorSync Manager in the `data` buffer. For each call, the ColorSync Manager specifies in the `size` parameter the number of bytes of data you should return. Each time your function is called it should read and return the requested data; it should also specify in the `size` parameter the actual number of bytes of data it returns.
- When the ColorSync Manager calls your function with the `cmCloseSpool` command, you should perform any required cleanup processes.

Version Notes
Starting in ColorSync version 2.5, the ColorSync Manager calls your function directly, without going through the preferred, or any, CMM.

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

Declared In
`CMTypes.h`

CMGetImageSpaceProcPtr

Defines a pointer to a function that obtains the signature of the data color space in which the color values of colors in an image are expressed.

```
typedef CMError (*CMGetImageSpaceProcPtr)
(
    const FSSpec * spec,
    OSType * space
);
```

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

```
CMError MyCMGetImageSpaceProc (
    const FSSpec * spec,
    OSType * space
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

space

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMGetIndImageProfileProcPtr

Defines a pointer to a function that obtains a specific embeded profile for a given image.

```
typedef CMError (*CMGetIndImageProfileProcPtr)
(
    const FSSpec * spec,
    UInt32 index,
    CMProfileRef * prof
);
```

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

```
CMError MyCMGetIndImageProfileProc (
    const FSSpec * spec,
    UInt32 index,
```

```
    CMProfileRef * prof
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

index

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMIterateDeviceInfoProcPtr

Defines a pointer to a function that iterates through device information available on the system.

```
typedef OSErr (*CMIterateDeviceInfoProcPtr)
(
    const CMDeviceInfo * deviceInfo,
    void * refCon
);
```

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

```
OSErr MyCMIterateDeviceInfoProc (
    const CMDeviceInfo * deviceInfo,
    void * refCon
);
```

Parameters

deviceData

refCon

Return Value

An `OSErr` value.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMDeviceIntegration.h`

CMIterateDeviceProfileProcPtr

Defines a pointer to a function that iterates through the device profiles available on the system.

```
typedef OSErr (*CMIterateDeviceProfileProcPtr)
(
    const CMDeviceInfo * deviceInfo,
    const NCMDeviceProfileInfo * profileInfo,
    void * refCon
);
```

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

```
OSErr MyCMIterateDeviceProfileProc (
    const CMDeviceInfo * deviceInfo,
    const NCMDeviceProfileInfo * profileInfo,
    void * refCon
);
```

Parameters

deviceData

profileData

refCon

Return Value
An `OSErr` value.

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

Declared In
`CMDeviceIntegration.h`

CMLinkImageProcPtr

Defines a pointer to a function that matches an image file with a device link profile.

```
typedef CMError (*CMLinkImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef lnkProf,
    UInt32 lnkIntent
);
```

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

```
CMError MyCMLinkImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef lnkProf,
    UInt32 lnkIntent
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl

qual

lnkProf

lnkIntent

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMMatchImageProcPtr

Defines a pointer to a function that color matches an image file.

```
typedef CMError (*CMMatchImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf
);
```

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

```
CMError MyCMMatchImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl
qual
srcProf
srcIntent
dstProf

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMMIterateProcPtr

Defines a pointer to a function that iterates through color management modules installed on the system.

```
typedef OSErr (*CMMIterateProcPtr) (
    CMMInfo * iterateData,
    void * refCon
);
```

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

```
OSErr MyCMMIterateProc (
    CMMInfo * iterateData,
    void * refCon
);
```

Parameters
iterateData
refCon

Return Value
An `OSErr` value.

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

Declared In
`CMApplication.h`

CMProfileAccessProcPtr

Defines a pointer to a profile access callback function that provides procedure-based access to a profile.

```
typedef OSErr (*CMProfileAccessProcPtr)
(
    SInt32 command,
    SInt32 offset,
    SInt32 *size,
    void *data,
    void *refCon
);
```

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

```
OSErr MyCMProfileAccessProc (
    SInt32 command,
    SInt32 offset,
    SInt32 *size,
    void *data,
    void *refCon
);
```

Parameters

command

A command value indicating the operation to perform. Operation constants are described in "Profile Access Procedures" (page 239).

offset

For read and write operations, the offset from the beginning of the profile at which to read or write data.

size

A pointer to a size value. On input, for the `cmReadAccess` and `cmWriteAccess` command constants, a pointer to a value indicating the number of bytes to read or write; for the `cmOpenWriteAccess` command, the total size of the profile. On return, after reading or writing, the actual number of bytes read or written.

data

A pointer to a buffer containing data to read or write. On return, for a read operation, contains the data that was read.

refCon

A reference constant pointer that can store private data for the `CMProfileAccessCallback` function.

Return Value

An `OSErr` value.

Discussion

When your application calls the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, or `CMNewLinkProfile` functions, it may supply the ColorSync Manager with a profile location structure of type `CMProfileLocation` (page 165) that specifies a procedure that provides access to a profile. In the structure, you provide a universal procedure pointer to a profile access procedure supplied by you and, optionally, a pointer to data your procedure can use. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed.

When the ColorSync Manager calls your profile access procedure, it passes a constant indicating the operation to perform. The operations include creating a new profile, reading from the profile, writing the profile, and so on. Operation constants are described in "Profile Access Procedures" (page 239). Your procedure must be able to respond to each of these constants.

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

Declared In
`CMTypes.h`

CMProfileFilterProcPtr

Defines a pointer to a profile filter callback function that examines the profile whose reference you specify and determines whether to include it in the profile search result list.

```
typedef Boolean (*CMProfileFilterProcPtr)
(
    CMProfileRef prof,
    void * refCon
);
```

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

```
Boolean MyCMProfileFilterProc (
    CMProfileRef prof,
    void * refCon
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile to test.

refCon

A pointer to a reference constant that holds data passed through from the `CMNewProfileSearch` function or the `CMUpdateProfileSearch` function.

Return Value
A value of `false` indicates that the profile should be included; `true` indicates that the profile should be filtered out.

Discussion
Your `MyCMProfileFilterCallback` function is called after the `CMNewProfileSearch` function searches for profiles based on the search record's contents as specified by the search bitmask.

When your application calls `CMNewProfileSearch`, it passes a reference to a search specification record of type `CMSearchRecord` of type `CMSearchRecord` (page 173) that contains a `filter` field. If the `filter` field contains a pointer to your `MyCMProfileFilterCallback` function, then your function is called to determine whether to exclude a profile from the search result list. Your function should return `true` for a given profile to exclude that profile from the search result list. If you do not want to filter profiles beyond the criteria in the search record, specify a `NULL` value for the search record's `filter` field.

After a profile has been included in the profile search result based on criteria specified in the search record, your `MyCMProfileFilterCallback` function can further examine the profile. For example, you may wish to include or exclude the profile based on criteria such as an element or elements not included in the `CMSearchRecord` search record. Your `MyCMProfileFilterCallback` function can also perform searching using `AND` or `OR` logic.

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

Declared In
CMTypes.h

CMProfileIterateProcPtr

Defines a pointer to a profile iteration callback function that the ColorSync Manager calls for each found profile file as it iterates over the available profiles.

```
typedef OSErr (*CMProfileIterateProcPtr)
(
    CMProfileIterateData * iterateData,
    void * refCon
);
```

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

```
OSErr MyCMProfileIterateProc (
    CMProfileIterateData * iterateData,
    void * refCon
);
```

Parameters

iterateData

A pointer to a structure of type CMProfileIterateData (page 164). When the function CMIterateColorSyncFolder (page 57) calls MyProfileIterateCallback, as it does once for each found profile, the structure contains key information about the profile.

refCon

An untyped pointer to arbitrary data your application previously passed to the function CMIterateColorSyncFolder (page 57).

Return Value
An OSErr value. If MyCMProfileIterateCallback returns an error, CMIterateColorSyncFolder stops iterating and returns the error value to its caller (presumably your code).

Discussion
When your application needs information about the profiles currently available in the profiles folder, it calls the function CMIterateColorSyncFolder (page 57) , which, depending on certain conditions, calls your callback routine once for each profile. See the description of CMIterateColorSyncFolder for information on when it calls the MyCMProfileIterateCallback function.

Your MyCMProfileIterateCallback function examines the structure pointed to by the iterateData parameter to obtain information about the profile it describes. The function determines whether to do anything with that profile, such as list its name in a pop-up menu of available profiles.

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

Declared In
CMApplication.h

CMProofImageProcPtr

Defines a pointer to a function that proofs an image.

```
typedef CMError (*CMProofImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf,
    CMProfileRef prfProf
);
```

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

```
CMError MyCMProofImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf,
    CMProfileRef prfProf
);
```

Parameters

specFrom
See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

repl

qual

srcProf

srcIntent

dstProf

prfProf

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMSetIndImageProfileProcPtr

Defines a pointer to a function that sets a specific embeded profile for a given image.

```
typedef CMError (*CMSetIndImageProfileProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 index,
    CMProfileRef prof
);
```

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

```
CMError MyCMSetIndImageProfileProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 index,
    CMProfileRef prof
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl

index

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMUnembedImageProcPtr

Defines a pointer to a function that umembeds an ICC profile from an image.

```
typedef CMError (*CMUnembedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl
);
```

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

```
CMError MyCMUnembedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl
```

```
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMValidImageProcPtr

Defines a pointer to a function that validates a specified image file.

```
typedef CMError (*CMValidImageProcPtr)
(
    const FSSpec * spec
);
```

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

```
CMError MyCMValidImageProc (
    const FSSpec * spec
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CountImageProfilesProcPtr

Defines a pointer to a function that counts the number of embeded profiles for a given image.

```
typedef CMError (*CountImageProfilesProcPtr)
(
    const FSSpec * spec,
    UInt32 * count
);
```

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

```
CMError MyCountImageProfilesProc (
    const FSSpec * spec,
    UInt32 * count
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

count

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

EmbedImageProcPtr

Defines a pointer to an embed-image function.

```
typedef CMError (*EmbedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    CMProfileRef embedProf,
    UInt32 embedFlags
);
```

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

```
CMError MyEmbedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    CMProfileRef embedProf,
    UInt32 embedFlags
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

embedProf
embedFlags

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

GetImageSpaceProcPtr

Defines a pointer to a get-image-space function.

```
typedef CMError (*GetImageSpaceProcPtr)
(
    const FSSpec * spec,
    OSType * space
);
```

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

```
CMError MyGetImageSpaceProc (
    const FSSpec * spec,
    OSType * space
);
```

Parameters

spec
See the File Manager documentation for a description of the `FSSpec` data type.

space

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

GetIndImageProfileProcPtr

Defines a pointer to a function that obtains a color profile for an individual image..

```
typedef CMError (*GetIndImageProfileProcPtr)
(
    const FSSpec * spec,
    UInt32 index,
    CMProfileRef * prof
);
```

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

```
CMError MyGetIndImageProfileProc (
    const FSSpec * spec,
    UInt32 index,
    CMProfileRef * prof
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

index

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

MatchImageProcPtr

Defines a pointer to a match-image function.

```
typedef CMError (*MatchImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 qual,
    UInt32 srcIntent,
    CMProfileRef srcProf,
    CMProfileRef dstProf,
    CMProfileRef prfProf,
    UInt32 matchFlags
);
```

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

```
CMError MyMatchImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 qual,
    UInt32 srcIntent,
    CMProfileRef srcProf,
    CMProfileRef dstProf,
```

```
    CMProfileRef prfProf,
    UInt32 matchFlags
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

qual

srcIntent

srcProf

dstProf

prfProf

matchFlags

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

SetIndImageProfileProcPtr

Defines a pointer to a function that sets a color profile for an individual image.

```
typedef CMError (*SetIndImageProfileProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 index,
    CMProfileRef prof,
    UInt32 embedFlags
);
```

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

```
CMError MySetIndImageProfileProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 index,
    CMProfileRef prof,
    UInt32 embedFlags
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

index

prof

embedFlags

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

UnembedImageProcPtr

Defines a pointer to an umembed-image function.

```
typedef CMError (*UnembedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto
);
```

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

```
CMError MyUnembedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto
);
```

Parameters

specFrom
See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

ValidateImageProcPtr

Defines a pointer to a validate-image function.

```
typedef CMError (*ValidateImageProcPtr)
(
    const FSSpec * spec
);
```

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

```
CMError MyValidateImageProc (
    const FSSpec * spec
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

ValidateSpaceProcPtr

Defines a pointer to a validate-space function.

```
typedef CMError (*ValidateSpaceProcPtr)
(
    const FSSpec * spec,
    OSType * space
);
```

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

```
CMError MyValidateSpaceProc (
    const FSSpec * spec,
    OSType * space
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

space

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

Data Types

CalibratorInfo

Contains data used to calibrate a display.

```
struct CalibratorInfo {
    UInt32 dataSize;
    CMDisplayIDType displayID;
    UInt32 profileLocationSize;
    CMProfileLocation * profileLocationPtr;
    CalibrateEventUPP eventProc;
    Boolean isGood;
};
typedef struct CalibratorInfo CalibratorInfo;
```

Fields

`dataSize`

`displayID`

`profileLocationSize`

`profileLocationPtr`

`eventProc`

`isGood`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMCalibrator.h`

CM2Header

Contains information that supports the header format specified by the ICC format specification for version 2.x profiles.

```
struct CM2Header {
    UInt32 size;
    OSType CMMType;
    UInt32 profileVersion;
    OSType profileClass;
    OSType dataColorSpace;
    OSType profileConnectionSpace;
    CMDateTime dateTime;
    OSType CS2profileSignature;
    OSType platform;
    UInt32 flags;
    OSType deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 renderingIntent;
    CMFixedXYZColor white;
    OSType creator;
    char reserved[44];
};
typedef struct CM2Header CM2Header;
```

Fields

`size`

The total size in bytes of the profile.

`CMMType`

The signature of the preferred CMM for color-matching and color-checking sessions for this profile. To avoid conflicts with other CMMs, this signature must be registered with the ICC. For the signature of the default CMM, see "Default CMM Signature" (page 218).

`profileVersion`

The version of the profile format. The first 8 bits indicate the major version number, followed by 8 bits indicating the minor version number. The following 2 bytes are reserved.

The profile version number is not tied to the version of the ColorSync Manager. Profile formats and their versions are defined by the ICC. For example, a major version change may indicate the addition of new required tags to the profile format; a minor version change may indicate the addition of new optional tags.

`profileClass`

One of the seven profile classes supported by the ICC: input, display, output, named color space, device link, color space conversion, or abstract. For the signatures representing profile classes, see "Profile Classes" (page 240).

`dataColorSpace`

The color space of the profile. Color values used to express colors of images using this profile are specified in this color space. For a list of the color space signatures, see "Color Space Signatures" (page 210).

`profileConnectionSpace`

The profile connection space, or PCS. The signatures for the two profile connection spaces supported by ColorSync, `cmXYZData` and `cmLabData`, are described in "Color Space Signatures" (page 210).

`dateTime`

The date and time when the profile was created. You can use this value to keep track of your own versions of this profile. For information on the date and time format, see `CMDateTime` (page 130).

`CS2profileSignature`

The `'acsp'` constant as required by the ICC format.

`platform`
: The signature of the primary platform on which this profile runs. For Apple Computer, this is `'APPL'`. For other platforms, refer to the International Color Consortium Profile Format Specification.

`flags`
: Flags that provide hints, such as preferred quality and speed options, to the preferred CMM. The `flags` field consists of an unsigned long data type. The 16 bits in the low word, 0-15, are reserved for use by the ICC. The 16 bits in the high word, 16-31, are available for use by color management systems. For information on how these bits are defined and how your application can set and test them, see "Flag Mask Definitions for Version 2.x Profiles" (page 224).

`deviceManufacturer`
: The signature of the manufacturer of the device to which this profile applies. This value is registered with the ICC.

`deviceModel`
: The model of this device, as registered with the ICC.

`deviceAttributes`
: Attributes that are unique to this particular device setup, such as media, paper, and ink types. The data type for this field is an array of two unsigned longs. The low word of `deviceAttributes[0]` is reserved by the ICC. The high word of `deviceAttributes[0]` and the entire word of `deviceAttributes[1]` are available for vendor use. For information on how the bits in `deviceAttributes` are defined and how your application can set and test them, see "Device Attribute Values for Version 2.x Profiles" (page 219).

`renderingIntent`
: The preferred rendering intent for the object or file tagged with this profile. Four types of rendering intent are defined: perceptual, relative colorimetric, saturation, and absolute colorimetric. The `renderingIntent` field consists of an unsigned long data type. The low word is reserved by the ICC and is used to set the rendering intent. The high word is available for use. For information on how the bits in `renderingIntent` are defined and how your application can set and test them, see "Rendering Intent Values for Version 2.x Profiles" (page 253).

`white`
: The profile illuminant white reference point, expressed in the XYZ color space.

`creator`
: Signature identifying the profile creator.

`reserved`
: This field is reserved for future use.

Discussion

The ColorSync Manager defines the `CM2header` profile structure to support the header format specified by the ICC format specification for version 2.x profiles. For a description of `CMHeader`, the ColorSync 1.0 profile header, see `CMHeader` (page 139). To obtain a copy of the International Color Consortium Profile Format Specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/.

Your application cannot obtain a discrete profile header value using the element tag scheme available for use with elements outside the header. Instead, to set or modify values of a profile header, your application must obtain the entire profile header using the function `CMGetProfileHeader` (page 47) and replace the header using the function `CMSetProfileHeader` (page 76).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CM2Profile

```
struct CM2Profile {
    CM2Header header;
    CMTagElemTable tagTable;
    char elemData[1];
};
typedef struct CM2Profile CM2Profile;
typedef CM2Profile * CM2ProfilePtr;
```

Fields

`header`

`tagTable`

`elemData`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CM4Header

```
struct CM4Header {
    UInt32 size;
    OSType CMMType;
    UInt32 profileVersion;
    OSType profileClass;
    OSType dataColorSpace;
    OSType profileConnectionSpace;
    CMDateTime dateTime;
    OSType CS2profileSignature;
    OSType platform;
    UInt32 flags;
    OSType deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 renderingIntent;
    CMFixedXYZColor white;
    OSType creator;
    CMProfileMD5 digest;
    char reserved[28];
};
typedef struct CM4Header CM4Header;
```

Fields

`size`
`CMMType`
`profileVersion`
`profileClass`
`dataColorSpace`
`profileConnectionSpace`
`dateTime`
`CS2profileSignature`
`platform`
`flags`
`deviceManufacturer`
`deviceModel`
`deviceAttributes`
`renderingIntent`
`white`
`creator`
`digest`
`reserved`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMICCProfile.h`

CMAccelerationCalcData

```
struct CMAccelerationCalcData {
    SInt32 pixelCount;
    Ptr inputData;
    Ptr outputData;
    UInt32 reserved1;
    UInt32 reserved2;
};
typedef struct CMAccelerationCalcData CMAccelerationCalcData;
```

Fields

CMAccelerationCalcDataPtr

```
typedef CMAccelerationCalcData* CMAccelerationCalcDataPtr;
```

CMAccelerationCalcDataHdl

```
typedef CMAccelerationCalcDataPtr* CMAccelerationCalcDataHdl;
```

CMAccelerationTableData

```
struct CMAccelerationTableData {
    SInt32 inputLutEntryCount;
    SInt32 inputLutWordSize;
    Handle inputLut;
    SInt32 outputLutEntryCount;
    SInt32 outputLutWordSize;
    Handle outputLut;
    SInt32 colorLutInDim;
    SInt32 colorLutOutDim;
    SInt32 colorLutGridPoints;
    SInt32 colorLutWordSize;
    Handle colorLut;
    CMBitmapColorSpace inputColorSpace;
    CMBitmapColorSpace outputColorSpace;
    void *userData;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
    UInt32 reserved4;
    UInt32 reserved5;
};
typedef struct CMAccelerationTableData CMAccelerationTableData;
```

Fields

CMAccelerationTableDataPtr

```
typedef CMAccelerationTableData* CMAccelerationTableDataPtr;
```

CMAccelerationTableDataHdl

```
typedef CMAccelerationTableDataPtr* CMAccelerationTableDataHdl;
```

CMAdaptationMatrixType

```
struct CMAdaptationMatrixType {
    OSType typeDescriptor;
    unsigned long reserved;
    Fixed adaptationMatrix[9];
};
typedef struct CMAdaptationMatrixType CMAdaptationMatrixType;
```

Fields

`typeDescriptor`

`reserved`

`adaptationMatrix`

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

Declared In
`CMICCProfile.h`

CMAppleProfileHeader

Defines a data structure to provide access to both version 2.x and version 1.0 profiles, as specified by the International Color Consortium.

```
union CMAppleProfileHeader {
    CMHeader cm1;
    CM2Header cm2;
    CM4Header cm4;
};
typedef union CMAppleProfileHeader CMAppleProfileHeader;
```

Fields

`cm1`

A version 1.0 profile header. For a description of the ColorSync version 1.0 profile header, see `CMHeader` (page 139).

`cm2`

A current profile header. For a description of the ColorSync profile header, see `CM2Header` (page 116).

`cm4`

Discussion
The ColorSync Manager defines the `CMAppleProfileHeader` structure to provide access to both version 2.x and version 1.0 profiles, as specified by the International Color Consortium. To obtain a copy of the International Color Consortium Profile Format Specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/.

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

Declared In
`CMApplication.h`

CMBitmap

Contains information that describes color bitmap images.

```
struct CMBitmap {
    char * image;
    long width;
    long height;
    long rowBytes;
    long pixelSize;
    CMBitmapColorSpace space;
    long user1;
    long user2;
};
typedef struct CMBitmap CMBitmap;
```

Fields

`image`
A pointer to a bit image.

`width`
The width of the bit image, that is, the number of pixels in a row.

`height`
The height of the bit image, that is, the number of rows in the image.

`rowBytes`
The offset in bytes from one row of the image to the next.

`pixelSize`
The number of bits per pixel. The pixel size should correspond to the packing size specified in the `space` field. This requirement is not enforced as of ColorSync version 2.5, but it may be enforced in future versions.

`space`
The color space in which the colors of the bitmap image are specified. For a description of the possible color spaces for color bitmaps, see "Color Space Constants With Packing Formats" (page 203).

`user1`
Not used by ColorSync. It is recommended that you set this field to 0.

`user2`
Not used by ColorSync. It is recommended that you set this field to 0.

Discussion
The ColorSync Manager defines a bitmap structure of type `CMBitmap` to describe color bitmap images. When your application calls the function `CWMatchColors` (page 87), you pass a pointer to a source bitmap of type `CMBitmap` containing the image whose colors are to be matched to the color gamut of the device specified by the destination profile of the given color world. If you do not want the image color matched in place, you can also pass a pointer to a resulting bitmap of type `CMBitmap` to define and hold the color-matched image.

For QuickDraw GX, an image can have an indexed bitmap to a list of colors. The ColorSync Manager does not support indexed bitmaps in the same way QuickDraw GX does. ColorSync supports indexed bitmaps only when the `cmNamedIndexed32Space` color space constant is used in conjunction with a named color space profile.

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

Declared In
`CMApplication.h`

CMBitmapCallBackProc

```
typedef CMBitmapCallBackProcPtr CMBitmapCallBackProc;
```

CMBitmapCallBackUPP

Defines a universal procedure pointer to a bitmap callback.

```
typedef CMBitmapCallBackProcPtr CMBitmapCallBackUPP;
```

Discussion
For more information, see the description of the `CMBitmapCallBackProcPtr` (page 93) callback function.

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

Declared In
`CMTypes.h`

CMBufferLocation

```
struct CMBufferLocation {
    void * buffer;
    UInt32 size;
};
typedef struct CMBufferLocation CMBufferLocation;
```

Fields
`buffer`
`size`

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

Declared In
`CMApplication.h`

CMCMYColor

Contains color values expressed in the CMY color space.

```
struct CMCMYColor {
    UInt16 cyan;
    UInt16 magenta;
    UInt16 yellow;
};
typedef struct CMCMYColor CMCMYColor;
```

Fields

`cyan`

`magenta`

`yellow`

Discussion

A color value expressed in the CMY color space is composed of `cyan`, `magenta`, and `yellow` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMCMYKColor

Contains color values expressed in the CMYK color space.

```
struct CMCMYKColor {
    UInt16 cyan;
    UInt16 magenta;
    UInt16 yellow;
    UInt16 black;
};
typedef struct CMCMYKColor CMCMYKColor;
```

Fields

`cyan`

`magenta`

`yellow`

`black`

Discussion

A color value expressed in the CMYK color space is composed of `cyan`, `magenta`, `yellow`, and `black` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMColor

Defines a union that can be used to specify a color value defined by one of the 15 data types supported by the union.

```
union CMColor {
    CMRGBColor rgb;
    CMHSVColor hsv;
    CMHLSColor hls;
    CMXYZColor XYZ;
    CMLabColor Lab;
    CMLuvColor Luv;
    CMYxyColor Yxy;
    CMCMYKColor cmyk;
    CMCMYColor cmy;
    CMGrayColor gray;
    CMMultichannel5Color mc5;
    CMMultichannel6Color mc6;
    CMMultichannel7Color mc7;
    CMMultichannel8Color mc8;
    CMNamedColor namedColor;
};
typedef union CMColor CMColor;
```

Fields

`rgb`
A color value expressed in the RGB color space as data of type `CMRGBColor` (page 171).

`hsv`
A color value expressed in the HSV color space as data of type `CMHSVColor` (page 142).

`hls`
A color value expressed in the HLS color space as data of type `CMHLSColor` (page 142).

`XYZ`
A color value expressed in the XYZ color space as data of type `CMXYZColor` (page 184).

`Lab`
A color value expressed in the L*a*b* color space as data of type `CMLabColor` (page 144).

`Luv`
A color value expressed in the L*u*v* color space as data of type `CMLuvColor` (page 146).

`Yxy`
A color value expressed in the Yxy color space as data of type `CMYxyColor` (page 185).

`cmyk`
A color value expressed in the CMYK color space as data of type `CMCMYKColor` (page 125).

`cmy`
A color value expressed in the CMY color space as data of type `CMCMYColor` (page 124).

`gray`
A color value expressed in the Gray color space as data of type `CMGrayColor` (page 138).

`mc5`
A color value expressed in the five-channel multichannel color space as data of type `CMMultichannel5Color`. See `CMMultichannel5Color` (page 151) for a description of the `CMMultichannel5Color` data type.

`mc6`
A color value expressed in the six-channel multichannel color space as data of type `CMMultichannel6Color`. See `CMMultichannel6Color` (page 152) for a description of the `CMMultichannel6Color` data type.

`mc7`
: A color value expressed in the seven-channel multichannel color space as data of type `CMMultichannel7Color`. See `CMMultichannel7Color` (page 152) for a description of the `CMMultichannel7Color` data type.

`mc8`
: A color value expressed in the eight-channel multichannel color space as data of type `CMMultichannel8Color`. See `CMMultichannel8Color` (page 152) for a description of the `CMMultichannel8Color` data type.

`namedColor`
: A color value expressed as an index into a named color space. See `CMNamedColor` (page 155) for a description of the `CMNamedColor` data type.

Discussion
A color union can contain one of the above fields.

Your application can use a union of type `CMColor` to specify a color value defined by one of the 15 data types supported by the union. Your application uses an array of color unions to specify a list of colors to match, check, or convert. The array is passed as a parameter to the general purpose color matching, color checking, or color conversion functions. The following functions use a color union:

- The function `CWMatchColors` (page 87) matches the colors in the color list array to the data color space of the destination profile specified by the color world.
- The function `CWCheckColors` (page 81) checks the colors in the color list array against the color gamut specified by the color world's destination profile.
- The color conversion functions, described in "Converting Between Color Spaces", take source and destination array parameters of type `CMColor` specifying lists of colors to convert from one color space to another.

You do not use a union of type `CMColor` to convert colors expressed in the XYZ color space as values of type `CMFixedXYZ` because the `CMColor` union does not support the `CMFixedXYZ` data type.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`CMApplication.h`

CMConcatCallBackUPP

Defines a universal procedure pointer to a progress-monitoring function that the ColorSync Manager calls during lengthy color world processing.

```
typedef CMConcatCallBackProcPtr CMConcatCallBackUPP;
```

Discussion
For more information, see the description of the `CMConcatCallBackProcPtr` (page 94) callback function.

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

Declared In
`CMTypes.h`

CMConcatProfileSet

Contains profile and other information needed to set up a color world.

```
struct CMConcatProfileSet {
    UInt16 keyIndex;
    UInt16 count;
    CMProfileRef profileSet[1];
};
typedef struct CMConcatProfileSet CMConcatProfileSet;
```

Fields

`keyIndex`

A zero-based index into the array of profile references identifying the profile whose CMM is used for the entire session. The profile's `CMMType` field identifies the CMM.

`count`

The one-based count of profiles in the profile array. A minimum of one profile is required.

`profileSet`

A variable-length array of profile references. The references must be in processing order from source to destination. The rules governing the types of profiles you can specify in a profile array differ depending on whether you are creating a profile set for the function `CWConcatColorWorld` (page 83) or for the function `CWNewLinkProfile` (page 310). See the function descriptions for details.

Discussion

You can call the function `NCWNewColorWorld` (page 90) to create a color world for operations such as color matching and color conversion. A color world is normally based on two profiles—source and destination. But it can include a series of profiles that describe the processing for a work-flow sequence, such as scanning, printing, and previewing an image. To create a color world that includes a series of profiles, you use the function `CWConcatColorWorld` (page 83).

The array specified in the `profileSet` field identifies a concatenated profile set your application can use to establish a color world in which the sequential relationship among the profiles exists until your application disposes of the color world. Alternatively, you can create a device link profile composed of a series of linked profiles that remains intact and available for use again after your application disposes of the concatenated color world. In either case, you use a data structure of type `CMConcatProfileSet` to define the profile set.

A device link profile accommodates users who use a specific configuration requiring a combination of device profiles and possibly non-device profiles repeatedly over time.

To set up a color world that includes a concatenated set of profiles, your application uses the function `CWConcatColorWorld` (page 83) , passing it a structure of type `CMConcatProfileSet` . The array you pass may contain a set of profile references or it may contain only the profile reference of a device link profile. To create a device link profile, your application calls the function `CWNewLinkProfile` (page 310), passing a structure of type `CMConcatProfileSet`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMCurveType

```
struct CMCurveType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 countValue;
    UInt16 data[1];
};
typedef struct CMCurveType CMCurveType;
```

Fields

`typeDescriptor`

`reserved`

`countValue`

`data`

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

Declared In
`CMICCProfile.h`

CMCWInfoRecord

Contains information about a given color world.

```
struct CMCWInfoRecord {
    UInt32 cmmCount;
    CMMInfoRecord cmmInfo[2];
};
typedef struct CMCWInfoRecord CMCWInfoRecord;
```

Fields

`cmmCount`

The number of CMMs involved in the color-matching session, either 1 or 2.

`cmmInfo`

An array containing two elements. Depending on the value that `cmmCount` returns, the `cmmInfo` array contains one or two records of type `CMMInfoRecord` (page 150) reporting the CMM type and version number.

If `cmmCount` is 1, the first element of the array (`cmmInfo[0]`) describes the CMM and the contents of the second element of the array (`cmmInfo[1]`) is undefined.

If `cmmCount` is 2, the first element of the array (`cmmInfo[0]`) describes the source CMM and the second element of the array (`cmmInfo[1]`) describes the destination CMM.

Discussion
Your application supplies a color world information record structure of type `CMCWInfoRecord` as a parameter to the `CMGetCWInfo` function to obtain information about a given color world. The ColorSync Manager uses this data structure to return information about the color world.

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

Not available to 64-bit applications.

Declared In
CMApplication.h

CMDataType

```
struct CMDataType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 dataFlag;
    char data[1];
};
typedef struct CMDataType CMDataType;
```

Fields
typeDescriptor
reserved
dataFlag
data

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

Declared In
CMICCProfile.h

CMDateTime

Contains data that specifies a date and time in year, month, day of the month, hours, minutes, and seconds

```
struct CMDateTime {
    UInt16 year;
    UInt16 month;
    UInt16 dayOfTheMonth;
    UInt16 hours;
    UInt16 minutes;
    UInt16 seconds;
};
typedef struct CMDateTime CMDateTime;
```

Fields
year
: The year. Note that to indicate the year 1984, this field would store the integer 1984, not just 84.

month
: The month of the year, where 1 represents January, and 12 represents December.

dayOfTheMonth
: The day of the month, ranging from 1 to 31.

hours
: The hour of the day, ranging from 0 to 23, where 0 represents midnight and 23 represents 11:00 P.M.

minutes
: The minutes of the hour, ranging from 0 to 59.

`seconds`
The seconds of the minute, ranging from 0 to 59.

Discussion
The ColorSync Manager defines the `CMDateTime` data structure to specify a date and time in year, month, day of the month, hours, minutes, and seconds. Other ColorSync structures use the `CMDateTime` structure to specify information such as the creation date or calibration date for a color space profile.

The `CMDateTime` structure is similar to the Macintosh Toolbox structure `DateTimeRec`, and like it, is intended to hold date and time values only for a Gregorian calendar.

The `CMDateTime` structure is platform independent. However, when used with Macintosh Toolbox routines such as `SecondsToDate` and `DateToSeconds`, which use seconds to designate years, the range of years that can be represented is limited.

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

Declared In
`CMICCProfile.h`

CMDateTimeType

```
struct CMDateTimeType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMDateTime dateTime;
};
typedef struct CMDateTimeType CMDateTimeType;
```

Fields
`typeDescriptor`
`reserved`
`dateTime`

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

Declared In
`CMICCProfile.h`

CMDeviceData

```
struct CMDeviceData {
    UInt32 dataVersion;
    CMDeviceSpec deviceSpec;
    CMDeviceScope deviceScope;
    CMDeviceState deviceState;
    CMDeviceProfileID defaultProfileID;
    UInt32 profileCount;
    UInt32 reserved;
};
typedef struct CMDeviceData CMDeviceData;
```

CMDeviceDataPtr

```
typedef CMDeviceData* CMDeviceDataPtr;
```

CMDeviceID

Defines a data type for a CM device ID.

```
typedef UInt32 CMDeviceID;
```

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

Declared In
`CMDeviceIntegration.h`

CMDeviceInfo

```
struct CMDeviceInfo {
    UInt32 dataVersion;
    CMDeviceClass deviceClass;
    CMDeviceID deviceID;
    CMDeviceScope deviceScope;
    CMDeviceState deviceState;
    CMDeviceProfileID defaultProfileID;
    CFDictionaryRef * deviceName;
    UInt32 profileCount;
    UInt32 reserved;
};
typedef struct CMDeviceInfo CMDeviceInfo;
typedef CMDeviceInfo * CMDeviceInfoPtr;
```

Fields

`dataVersion`

`deviceClass`

`deviceID`

`deviceScope`

`deviceState`

`defaultProfileID`

`deviceName`

See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

`profileCount`

`reserved`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMDeviceIntegration.h`

CMDeviceName

```
struct CMDeviceName {
    UniCharCount deviceNameLength;
    UniChar deviceName[256];
};
typedef struct CMDeviceName CMDeviceName;
```

Fields

CMDeviceNamePtr

```
typedef CMDeviceName* CMDeviceNamePtr;
```

CMDeviceProfileArray

```
struct CMDeviceProfileArray {
    UInt32 profileCount;
    CMDeviceProfileInfo profiles[1];
};
typedef struct CMDeviceProfileArray CMDeviceProfileArray;
typedef CMDeviceProfileArray * CMDeviceProfileArrayPtr;
```

Fields

`profileCount`

`profiles`

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

Declared In
`CMDeviceIntegration.h`

CMDeviceProfileID

```
typedef UInt32 CMDeviceProfileID;
```

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

Declared In
`CMDeviceIntegration.h`

CMDeviceProfileInfo

```
struct CMDeviceProfileInfo {
    UInt32 dataVersion;
    CMDeviceProfileID profileID;
    CMProfileLocation profileLoc;
    CFDictionaryRef profileName;
    UInt32 reserved;
};
typedef struct CMDeviceProfileInfo CMDeviceProfileInfo;
```

Fields

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

Declared In
`CMDeviceIntegration.h`

CMDeviceProfileScope

```
typedef CMDeviceScope CMDeviceProfileScope;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMDeviceScope

```
struct CMDeviceScope {
    CFStringRef deviceUser;
    CFStringRef deviceHost;
};
typedef struct CMDeviceScope CMDeviceScope;
typedef CMDeviceScope CMDeviceProfileScope;
```

Fields
`deviceUser`
`deviceHost`

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`CMDeviceIntegration.h`

CMDeviceSpec

```
struct CMDeviceSpec {
    UInt32 specVersion;
    CMDeviceClass deviceClass;
    CMDeviceID deviceID;
    CMDeviceName deviceName;
    UInt32 reserved;
};
typedef struct CMDeviceSpec CMDeviceSpec;
```

Fields

CMDeviceSpecPtr

```
typedef CMDeviceSpec* CMDeviceSpecPtr;
```

CMDeviceState

```
typedef UInt32 CMDeviceState;
```

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

Declared In
CMDeviceIntegration.h

CMDisplayIDType

Defines a data type for a display ID type.

```
typedef UInt32 CMDisplayIDType;
```

Discussion
This data type is passed as a parameter to the functions `CMGetProfileByAVID` (page 44) and `CMSetProfileByAVID` (page 72).

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

Declared In
CMTypes.h

CMError

Defines a data type for a ColorSync Manager result code.

```
typedef  CMError;
```

Discussion
For a list of possible resutlt codes, see "ColorSync Manager Result Codes" (page 261).

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

Declared In
`CMTypes.h`

CMFileLocation

Contains a file specification for a profile stored in a disk file.

```
struct CMFileLocation {
    FSSpec spec;
};
typedef struct CMFileLocation CMFileLocation;
```

Fields

`spec`

A file system specification structure giving the location of the profile file. A file specification structure includes the volume reference number, the directory ID of the parent directory, and the filename or directory name. See the File Manager documentation for a description of the `FSSpec` data type.

Discussion
Your application uses the `CMFileLocation` structure to provide a file specification for a profile stored in a disk file. You provide a file specification structure in the `CMProfileLocation` structure's `u` field to specify the location of an existing profile or a profile to be created.

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

Declared In
`CMApplication.h`

CMFixedXYColor

```
struct CMFixedXYColor {
    Fixed x;
    Fixed y;
};
typedef struct CMFixedXYColor CMFixedXYColor;
```

Fields

`x`

`y`

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CMICCProfile.h`

CMFixedXYZColor

Contains data that specifies the profile illuminant in the profile header's `white` field and other profile element values.

```
struct CMFixedXYZColor {
    Fixed X;
    Fixed Y;
    Fixed Z;
};
typedef struct CMFixedXYZColor CMFixedXYZColor;
```

Fields

`X`

`Y`

`Z`

Discussion

ColorSync uses the `CMFixedXYZColor` data type to specify the profile illuminant in the profile header's `white` field and to specify other profile element values. Color component values defined by the `Fixed` type definition can be used to specify a color value in the XYZ color space with greater precision than a color whose components are expressed as `CMXYZComponent` data types. The `Fixed` data type is a signed 32-bit value. A color value expressed in the XYZ color space whose color components are of type `Fixed` is defined by the `CMFixedXYZColor` type definition.

Your application can convert colors defined in the XYZ color space between `CMXYZColor` data types (in which the color components are expressed as 16-bit unsigned values) and `CMFixedXYZColor` data types (in which the colors are expressed as 32-bit signed values). To convert color values, you use the functions `CMConvertFixedXYZToXYZ` (page 273) and `CMConvertXYZToFixedXYZ` (page 280).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMFlattenUPP

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

```
typedef CMFlattenProcPtr CMFlattenUPP;
```

Discussion

For more information, see the description of the `CMFlattenProcPtr` (page 96) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMTypes.h`

CMGrayColor

Contains a color value expressed in the gray color space.

```
struct CMGrayColor {
    UInt16 gray;
};
typedef struct CMGrayColor CMGrayColor;
```

Fields
`gray`

Discussion
A color value expressed in the Gray color space is composed of a single component, `gray`, represented as a numeric value within the range of 0 to 65535 inclusive.

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

Declared In
`CMApplication.h`

CMHandleLocation

Contains a handle specification for a profile stored in relocatable memory.

```
struct CMHandleLocation {
    Handle h;
};
typedef struct CMHandleLocation CMHandleLocation;
```

Fields
`h`
A data structure of type `Handle` containing a handle that indicates the location of a profile in memory.

Discussion
Your application uses the `CMHandleLocation` structure to provide a handle specification for a profile stored in relocatable memory. You provide the handle specification structure in the `CMProfileLocation` structure's `u` field to specify an existing profile or a profile to be created.

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

Declared In
`CMApplication.h`

CMHeader

Contains version 1.0 profile header data.

```
struct CMHeader {
    UInt32 size;
    OSType CMMType;
    UInt32 applProfileVersion;
    OSType dataType;
    OSType deviceType;
    OSType deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 profileNameOffset;
    UInt32 customDataOffset;
    CMMatchFlag flags;
    CMMatchOption options;
    CMXYZColor white;
    CMXYZColor black;
};
typedef struct CMHeader CMHeader;
```

Fields

`size`
: The total size in bytes of the profile, including any custom data.

`CMMType`
: The signature of the preferred CMM for color-matching and color-checking sessions for this profile. To avoid conflicts with other CMMs, this signature must be registered with the ICC. For the signature of the default CMM, see "Default CMM Signature" (page 218).

`applProfileVersion`
: The Apple profile version. Set this field to $0100 (defined as the constant `kCMApplProfileVersion`).

`dataType`
: The kind of color data.

`deviceType`
: The kind of device.

`deviceManufacturer`
: A name supplied by the device manufacturer.

`deviceModel`
: The device model specified by the manufacturer.

`deviceAttributes`
: Private information such as paper surface and ink temperature.

`profileNameOffset`
: The offset to the profile name from the top of data.

`customDataOffset`
: The offset to any custom data from the top of data.

`flags`

A field used by drivers; it can hold one of the following flags:

`CMNativeMatchingPreferredCMTurnOffCache`

The `CMNativeMatchingPreferred` flag is available for developers of intelligent peripherals that can off-load color matching into the peripheral. Most drivers will not use this flag. (Its default setting is 0, meaning that the profile creator does not care whether matching occurs on the host or the device.)

Use the `CMTurnOffCache` flag for CMMs that will not benefit from a cache, such as those that can look up data from a table with less overhead, or that do not want to take the memory hit a cache entails, or that do their own caching and do not want the CMM to do it. (The default is 0, meaning turn on cache.

`options`

The `options` field specifies the preferred matching for this profile; the default is `CMPerceptualMatch`; other values are `CMColorimetricMatch` or `CMSaturationMatch`. The options are set by the image creator.

`white`

The profile illuminant white reference point, expressed in the XYZ color space.

`black`

The black reference point for this profile, expressed in the XYZ color space.

Discussion

ColorSync 1.0 defined a version 1.0 profile whose structure and format are different from that of the ICC version 2.x profile. The `CMHeader` data type represents the version 1.0 profile header. For more information on profile version numbers, see "ColorSync and ICC Profile Format Version Numbers". To obtain a copy of the International Color Consortium Profile Format Specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/

Your application cannot use ColorSync Manager functions to update a version 1.0 profile or to search for version 1.0 profiles. However, your application can use other ColorSync Manager functions that operate on version 1.0 profiles. For example, your application can open a version 1.0 profile using the function `CMOpenProfile` (page 63), obtain the version 1.0 profile header using the function `CMGetProfileHeader` (page 47), and access version 1.0 profile elements using the function `CMGetProfileElement` (page 46).

To make it possible to operate on both version 1.0 profiles and version 2.x profiles, the ColorSync Manager defines the union `CMAppleProfileHeader` (page 122), which supports either profile -*header version. The `CMHeader` data type defines the version 1.0 profile header, while the `CM2Header` (page 116) data type defines the version 2.x profile header.

Version Notes

Use of the `CMHeader` type is not recommended for ColorSync versions starting with 2.0. Use `CM2Header` (page 116) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMHLSColor

Contains a color value expressed in the HLS color space.

```
struct CMHLSColor {
    UInt16 hue;
    UInt16 lightness;
    UInt16 saturation;
};
typedef struct CMHLSColor CMHLSColor;
```

Fields

`hue`
A hue value that represents a fraction of a circle in which red is positioned at 0. .

`lightness`
A lightness value.

`saturation`
A saturation value.

Discussion

A color value expressed in the HLS color space is composed of `hue`, `lightness`, and `saturation` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMHSVColor

Contains a color value expressed in the HSV color space.

```
struct CMHSVColor {
    UInt16 hue;
    UInt16 saturation;
    UInt16 value;
};
typedef struct CMHSVColor CMHSVColor;
```

Fields

`hue`

`saturation`

`value`

Discussion

A color value expressed in the HSV color space is composed of `hue`, `saturation`, and `value` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive. The `hue` value represents a fraction of a circle in which red is positioned at 0.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMIntentCRDVMSize

Defines the rendering intent and its maximum VM size.

```
struct CMIntentCRDVMSize {
    long renderingIntent;
    UInt32 VMSize;
};
typedef struct CMIntentCRDVMSize CMIntentCRDVMSize;
```

Fields

`renderingIntent`

The rendering intent whose CRD virtual memory size you want to obtain. The rendering intent values are described in "Rendering Intent Values for Version 2.x Profiles" (page 253).

`VMSize`

The virtual memory size of the CRD for the rendering intent specified for the `renderingIntent` field.

Discussion

To specify the maximum virtual memory (VM) size of the color rendering dictionary (CRD) for a specific rendering intent for a particular PostScript(TM) Level 2 printer type, a printer profile can include the optional Apple-defined `'psvm'` tag. The PostScript CRD virtual memory size tag structure's element data includes an array containing one entry for each rendering intent and its virtual memory size.

If a PostScript printer profile includes this tag, the default CMM uses the tag and returns the values specified by the tag when your application or device driver calls the function `CMGetPS2ColorRenderingVMSize` (page 52).

If a PostScript printer profile does not include this tag, the CMM uses an algorithm to determine the VM size of the CRD. This may result in a size that is greater than the actual VM size.

The `CMPS2CRDVMSizeType` data type for the tag includes an array containing one or more members of type `CMIntentCRDVMSize`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMIString

Defines a profile name.

```
struct CMIString {
    ScriptCode theScript;
    Str63 theString;
};
typedef struct CMIString CMIString;
typedef CMIString IString;
```

Fields

`theScript`

The script code for the `theString` parameter.

`theString`

The profile name.

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

Declared In
`CMICCProfile.h`

CMLabColor

Contains a color value expressed in the L*a*b* color space.

```
struct CMLabColor {
    UInt16 L;
    UInt16 a;
    UInt16 b;
};
typedef struct CMLabColor CMLabColor;
```

Fields

`L`

A numeric value within the range of 0 to 65535, which maps to 0 to 100 inclusive. Note that this encoding is slightly different from the 0 to 65280 encoding of the L channel defined in the ICC specification for PCS L*a*b values.

`a`

A value that ranges from 0 to 65535, and maps to –128 to 127.996 inclusive.

`b`

A value that ranges from 0 to 65535, and maps to –128 to 127.996 inclusive.

Discussion
A color expressed in the L*a*b* color space is composed of `L`, `a`, and `b` component values.

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

Declared In
`CMApplication.h`

CMLut16Type

```
struct CMLut16Type {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 inputChannels;
    UInt8 outputChannels;
    UInt8 gridPoints;
    UInt8 reserved2;
    Fixed matrix[3][3];
    UInt16 inputTableEntries;
    UInt16 outputTableEntries;
    UInt16 inputTable[1];
};
typedef struct CMLut16Type CMLut16Type;
```

Fields

`typeDescriptor`

`reserved`

`inputChannels`

`outputChannels`

`gridPoints`

`reserved2`

`matrix`

`inputTableEntries`

`outputTableEntries`

`inputTable`

`CLUT`

`outputTable`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMLut8Type

```
struct CMLut8Type {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 inputChannels;
    UInt8 outputChannels;
    UInt8 gridPoints;
    UInt8 reserved2;
    Fixed matrix[3][3];
    UInt8 inputTable[1];
};
typedef struct CMLut8Type CMLut8Type;
```

Fields

`typeDescriptor`
`reserved`
`inputChannels`
`outputChannels`
`gridPoints`
`reserved2`
`matrix`
`inputTable`
`CLUT`
`outputTable`
`aNet`
`aNode`
`aSocket`

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

Declared In
`CMICCProfile.h`

CMLuvColor

Contains a color value expressed in the L*u*v* color space.

```
struct CMLuvColor {
    UInt16 L;
    UInt16 u;
    UInt16 v;
};
typedef struct CMLuvColor CMLuvColor;
```

Fields

`L`
: A numeric value within the range of 0 to 65535 that maps to 0 to 100 inclusive.

`u`
: A numeric value within the range of 0 to 65535 that maps to –128 to 127.996 inclusive.

`v`
: A numeric value within the range of 0 to 65535 that maps to –128 to 127.996 inclusive.

Discussion
A color value expressed in the L*u*v* color space is composed of `L`, `u`, and `v` component values.

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

Declared In
`CMApplication.h`

CMMakeAndModel

Contains make and model information fro a device.

```
struct CMMakeAndModel {
    OSType manufacturer;
    UInt32 model;
    UInt32 serialNumber;
    UInt32 manufactureDate;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
    UInt32 reserved4;
};
typedef struct CMMakeAndModel CMMakeAndModel;
```

Fields
`manufacturer`
`model`
`serialNumber`
`manufactureDate`
`reserved1`
`reserved2`
`reserved3`
`reserved4`

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

Declared In
`CMICCProfile.h`

CMMakeAndModelType

Contains make and model information along with a type descriptor.

```
struct CMMakeAndModelType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMMakeAndModel makeAndModel;
};
typedef struct CMMakeAndModelType CMMakeAndModelType;
```

Fields

`typeDescriptor`

`reserved`

`makeAndModel`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMMatchFlag

Defines a data type for match flags.

```
typedef  long CMMatchFlag;
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMMatchOption

Defines a data type for match options.

```
typedef long CMMatchOption;
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMMatchRef

Defines an abstract private data structure for the color-matching-session reference.

```
typedef struct OpaqueCMMatchRef * CMMatchRef;
```

Discussion
The ColorSync Manager defines an abstract private data structure of type `OpaqueCMMatchRef` for the color-matching-session reference. When your application calls the function `NCMBeginMatching` (page 269) to begin a QuickDraw-specific color-matching session, the ColorSync Manager returns a reference pointer to the color-matching session which you must later pass to the `CMEndMatching` function to conclude the session.

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

Declared In
`CMTypes.h`

CMMeasurementType

Contains measurement type information.

```
struct CMMeasurementType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 standardObserver;
    CMFixedXYZColor backingXYZ;
    UInt32 geometry;
    UInt32 flare;
    UInt32 illuminant;
};
typedef struct CMMeasurementType CMMeasurementType;
```

Fields
`typeDescriptor`
`reserved`
`standardObserver`
`backingXYZ`
`geometry`
`flare`
`illuminant`

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

Declared In
`CMICCProfile.h`

CMMInfo

Contains information pertainting to a color management module.

```
struct CMMInfo {
    UInt32 dataSize;
    OSType CMMType;
    OSType CMMMfr;
    UInt32 CMMVersion;
    unsigned char ASCIIName[32];
    unsigned char ASCIIDesc[256];
    UniCharCount UniCodeNameCount;
    UniChar UniCodeName[32];
    UniCharCount UniCodeDescCount;
    UniChar UniCodeDesc[256];
};
typedef struct CMMInfo CMMInfo;
```

Fields

`dataSize`

`CMMType`

`CMMMfr`

`CMMVersion`

`ASCIIName`

`ASCIIDesc`

`UniCodeNameCount`

`UniCodeName`

`UniCodeDescCount`

`UniCodeDesc`

`TPLFMT_BKSZ`

`TPLFMT_NBLOCKS`

`TPLFMT_EDCLOC`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMMInfoRecord

Contains CMM type and version information.

```
struct CMMInfoRecord {
    OSType CMMType;
    long CMMVersion;
};
typedef struct CMMInfoRecord CMMInfoRecord;
```

Fields

`CMMType`

The signature of the CMM as specified in the profile header's `CMMType` field. The `CMGetCWInfo` function returns this value.

`CMMVersion`

The version of the CMM. The `CMGetCWInfo` function returns this value.

Discussion
Your application supplies an array containing two CMM information record structures of type `CMMInfoRecord` as a field of the `CMCWInfoRecord` structure. These structures allow the `CMGetCWInfo` function to return information about the one or two CMMs used in a given color world. Your application must allocate memory for the array. When your application calls the `CMGetCWInfo` function, it passes a pointer to the `CMCWInfoRecord` structure containing the array.

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

Declared In
`CMApplication.h`

CMMIterateUPP

Defines a universal procedure pointer to a CMM interation callback.

```
typedef CMMIterateProcPtr CMMIterateUPP;
```

Discussion
For more information, see the description of the `CMMIterateProcPtr` (page 103) callback function.

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

Declared In
`CMApplication.h`

CMMultichannel5Color

Contains a color value expressed in the multichannel color space with 5 channels.

```
struct CMMultichannel5Color {
    UInt8 components[5];
};
typedef struct CMMultichannel5Color CMMultichannel5Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 5 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

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

Declared In
`CMApplication.h`

CMMultichannel6Color

Contains a color expressed in the multichannel color space with 6 channels.

```
struct CMMultichannel6Color {
    UInt8 components[6];
};
typedef struct CMMultichannel6Color CMMultichannel6Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 6 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

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

Declared In
`CMApplication.h`

CMMultichannel7Color

Contains a color value expressed in the multichannel color space with 7 channels.

```
struct CMMultichannel7Color {
    UInt8 components[7];
};
typedef struct CMMultichannel7Color CMMultichannel7Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 7 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

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

Declared In
`CMApplication.h`

CMMultichannel8Color

Contains a color value expressed in the multichannel color space with 8 channels

```
struct CMMultichannel8Color {
    UInt8 components[8];
};
typedef struct CMMultichannel8Color CMMultichannel8Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 8 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

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

Declared In
`CMApplication.h`

CMMultiFunctCLUTType

```
struct CMMultiFunctCLUTType {
    UInt8 gridPoints[16];
    UInt8 entrySize;
    UInt8 reserved[3];
    UInt8 data[1];
};
typedef struct CMMultiFunctCLUTType CMMultiFunctCLUTType;
```

Fields
`gridPoints`
`entrySize`
`reserved`
`data`

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CMICCProfile.h`

CMMultiFunctLutA2BType

```
typedef CMMultiFunctLutType CMMultiFunctLutA2BType;
```

Availability
Available in Mac OS X v10.1 through Mac OS X v10.4.

Declared In
`CMICCProfile.h`

CMMultiFunctLutB2AType

```
typedef CMMultiFunctLutType CMMultiFunctLutB2AType;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
CMICCProfile.h

CMMultiFunctLutType

```
struct CMMultiFunctLutType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 inputChannels;
    UInt8 outputChannels;
    UInt16 reserved2;
    UInt32 offsetBcurves;
    UInt32 offsetMatrix;
    UInt32 offsetMcurves;
    UInt32 offsetCLUT;
    UInt32 offsetAcurves;
    UInt8 data[1];
};
typedef struct CMMultiFunctLutType CMMultiFunctLutType;
typedef CMMultiFunctLutType CMMultiFunctLutA2BType;
```

Fields
typeDescriptor
reserved
inputChannels
outputChannels
reserved2
offsetBcurves
offsetMatrix
offsetMcurves
offsetCLUT
offsetAcurves
data

Availability
Available in Mac OS X v10.1 through Mac OS X v10.4.

Declared In
CMICCProfile.h

CMMultiLocalizedUniCodeEntryRec

```
struct CMMultiLocalizedUniCodeEntryRec {
    char languageCode[2];
    char regionCode[2];
    UInt32 textLength;
    UInt32 textOffset;
};
typedef struct CMMultiLocalizedUniCodeEntryRec CMMultiLocalizedUniCodeEntryRec;
```

Fields

`languageCode`

`regionCode`

`textLength`

`textOffset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMMultiLocalizedUniCodeType

```
struct CMMultiLocalizedUniCodeType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 entryCount;
    UInt32 entrySize;
};
typedef struct CMMultiLocalizedUniCodeType CMMultiLocalizedUniCodeType;
```

Fields

`typeDescriptor`

`reserved`

`entryCount`

`entrySize`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMNamedColor

Contains a color value expressed in a named color space.

```
struct CMNamedColor {
    UInt32 namedColorIndex;
};
typedef struct CMNamedColor CMNamedColor;
```

Fields
`namedColorIndex`

Discussion
A color value expressed in a named color space is composed of a single component, `namedColorIndex`, represented as a numeric value within the range of an unsigned long, or 1 to 232 – 1 inclusive.

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

Declared In
`CMApplication.h`

CMNamedColor2EntryType

```
struct CMNamedColor2EntryType {
    UInt8 rootName[32];
    UInt16 PCSColorCoords[3];
    UInt16 DeviceColorCoords[1];
};
typedef struct CMNamedColor2EntryType CMNamedColor2EntryType;
```

Fields
`rootName`
`PCSColorCoords`
`DeviceColorCoords`

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

Declared In
`CMICCProfile.h`

CMNamedColor2Type

```
struct CMNamedColor2Type {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 vendorFlag;
    UInt32 count;
    UInt32 deviceChannelCount;
    UInt8 prefixName[32];
    UInt8 suffixName[32];
    char data[1];
};
typedef struct CMNamedColor2Type CMNamedColor2Type;
```

Fields

`typeDescriptor`

`reserved`

`vendorFlag`

`count`

`deviceChannelCount`

`prefixName`

`suffixName`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMNamedColorType

```
struct CMNamedColorType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 vendorFlag;
    UInt32 count;
    UInt8 prefixName[1];
};
typedef struct CMNamedColorType CMNamedColorType;
```

Fields

`typeDescriptor`

`reserved`

`vendorFlag`

`count`

`prefixName`

`suffixName`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMNativeDisplayInfo

Contains color information for a native display.

```
struct CMNativeDisplayInfo {
    UInt32 dataSize;
    CMFixedXYColor redPhosphor;
    CMFixedXYColor greenPhosphor;
    CMFixedXYColor bluePhosphor;
    CMFixedXYColor whitePoint;
    Fixed redGammaValue;
    Fixed greenGammaValue;
    Fixed blueGammaValue;
    UInt16 gammaChannels;
    UInt16 gammaEntryCount;
    UInt16 gammaEntrySize;
    char gammaData[1];
};
typedef struct CMNativeDisplayInfo CMNativeDisplayInfo;
```

Fields
dataSize
redPhosphor
greenPhosphor
bluePhosphor
whitePoint
redGammaValue
greenGammaValue
blueGammaValue
gammaChannels
gammaEntryCount
gammaEntrySize
gammaData

Availability
Available in Mac OS X v10.1 and later.

Declared In
CMICCProfile.h

CMNativeDisplayInfoType

Contins color information and a tpe descriptor for a native display.

```
struct CMNativeDisplayInfoType {
    OSType typeDescriptor;
    unsigned long reserved;
    CMNativeDisplayInfo nativeDisplayInfo;
};
typedef struct CMNativeDisplayInfoType CMNativeDisplayInfoType;
```

Fields

`typeDescriptor`

`reserved`

`nativeDisplayInfo`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMICCProfile.h`

CMParametricCurveType

```
struct CMParametricCurveType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt16 functionType;
    UInt16 reserved2;
    Fixed value[1];
};
typedef struct CMParametricCurveType CMParametricCurveType;
```

Fields

`typeDescriptor`

`reserved`

`functionType`

`reserved2`

`value`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMICCProfile.h`

CMPathLocation

Contains path information.

```
struct CMPathLocation {
    char path[256];
};
typedef struct CMPathLocation CMPathLocation;
```

Fields

`path`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMProcedureLocation

Contains a universal procedure pointer to a profile access procedure.

```
struct CMProcedureLocation {
    CMProfileAccessUPP proc;
    void * refCon;
};
typedef struct CMProcedureLocation CMProcedureLocation;
```

Fields

`proc`

A universal procedure pointer to a profile access procedure. For a description of the procedure, see the function `CMProfileAccessProcPtr` (page 103).

`refCon`

A pointer to the profile access procedure's private data, such as a file or resource name, a pointer to a current offset, and so on.

Discussion

Your application uses the `CMProcedureLocation` structure to provide a universal procedure pointer to a profile access procedure. You provide this structure in the `CMProfileLocation` structure's `u` field. The `CMProcedureLocation` structure also contains a pointer field to specify data associated with the profile access procedure.

The ColorSync Manager calls your profile access procedure when the profile is created, initialized, opened, read, updated, or closed.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMProfile

```
struct CMProfile {
    CMHeader header;
    CMProfileChromaticities profile;
    CMProfileResponse response;
    CMIString profileName;
    char customData[1];
};
typedef struct CMProfile CMProfile;
typedef CMProfile * CMProfilePtr;
```

Fields

`header`

`profile`

`response`

`profileName`

`customData`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMProfileAccessUPP

Defines a universal procedure pointer to a profile access callback.

```
typedef CMProfileAccessProcPtr CMProfileAccessUPP;
```

Discussion

For more information, see the description of the `CMProfileAccessProcPtr` (page 103)) callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMTypes.h`

CMProfileChromaticities

```
struct CMProfileChromaticities {
    CMXYZColor red;
    CMXYZColor green;
    CMXYZColor blue;
    CMXYZColor cyan;
    CMXYZColor magenta;
    CMXYZColor yellow;
};
typedef struct CMProfileChromaticities CMProfileChromaticities;
```

Fields

`red`

`green`

`blue`

`cyan`

`magenta`

`yellow`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMProfileFilterProc

```
typedef CMProfileFilterProcPtr CMProfileFilterProc;
```

CMProfileFilterUPP

Defines a universal procedure pointer to a profile filter callback.

```
typedef CMProfileFilterProcPtr CMProfileFilterUPP;
```

Discussion

For more information, see the description of the `CMProfileFilterProcPtr` (page 105) callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMTypes.h`

CMProfileIdentifier

Contains data that can identify a profile but that takes up much less space than a large profile.

```
struct CMProfileIdentifier {
    CM2Header profileHeader;
    CMDateTime calibrationDate;
    UInt32 ASCIIProfileDescriptionLen;
    char ASCIIProfileDescription[1];
};
typedef struct CMProfileIdentifier CMProfileIdentifier;
typedef CMProfileIdentifier * CMProfileIdentifierPtr;
```

Fields

`profileHeader`

A version 2.x profile header structure. For more information, see `CM2Header` (page 116). In determining a profile match, all header fields are considered, except for primary platform, flags, and rendering intent.

`calibrationDate`

A structure of type `CMDateTime` (page 130), which specifies year, month, day of month, hours, minutes, and seconds. This field is optional—when set to 0, it is not considered in determining a profile match. When nonzero, it is compared to the `'calt'` tag data.

`ASCIIProfileDescriptionLen`

The length of the ASCII description string that follows.

`ASCIIProfileDescription`

The ASCII profile description string, as specified by the profile description tag.

Discussion

Embedding a profile in an image guarantees that the image can be rendered correctly on a different system. However, profiles can be large—as much as several hundred kilobytes. The ColorSync Manager defines a profile identifier structure, `CMProfileIdentifier`, that can identify a profile but that takes up much less space than a large profile.

The profile identifier structure contains a profile header, an optional calibration date, a profile description string length, and a variable-length profile description string. Your application might use an embedded profile identifier, for example, to change just the rendering intent or the flag values in an image without having to embed an entire copy of a profile. Rendering intent is described in "Rendering Intent Values for Version 2.x Profiles" (page 253) and flag values are described in "Flag Mask Definitions for Version 2.x Profiles" (page 224).

A document containing an embedded profile identifier cannot necessarily be ported to different systems or platforms.

The ColorSync Manager provides the function routine `NCMUseProfileComment` (page 272) to embed profiles and profile identifiers in an open picture file. Your application can embed profile identifiers in place of entire profiles, or in addition to them. A profile identifier can refer to an embedded profile or to a profile on disk.

The ColorSync Manager provides two routines for finding a profile identifier:

- `CMProfileIdentifierListSearch` (page 299) for finding a profile identifier in a list of profile identifiers
- `CMProfileIdentifierFolderSearch` (page 298) for finding a profile identifier in the ColorSync Profiles folder.

The descriptions of those functions provide information on searching algorithms. See also `CMProfileSearchRef` (page 168)

The `CMProfileIdentifierPtr` type definition defines a pointer to a profile identifier structure.

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

Declared In
`CMApplication.h`

CMProfileIterateData

Contains a callback routine with a description of a profile that is during an iteration through the available profiles.

```
struct CMProfileIterateData {
    UInt32 dataVersion;
    CM2Header header;
    ScriptCode code;
    Str255 name;
    CMProfileLocation location;
    UniCharCount uniCodeNameCount;
    UniChar * uniCodeName;
    unsigned char * asciiName;
    CMMakeAndModel * makeAndModel;
    CMProfileMD5 * digest;
};
typedef struct CMProfileIterateData CMProfileIterateData;
```

Fields

`dataVersion`
A value identifying the version of the structure. Currently set to `cmProfileIterateDataVersion1`.

`header`
A ColorSync version 2.x profile header structure of type `CM2Header` (page 116), containing information such as the profile size, type, version, and so on.

`code`
A script code identifying the script system used for the profile description. The `ScriptCode` data type is defined in the MacTypes.h header file.

`name`
The profile name, stored as a Pascal-type string (with length byte first) of up to 255 characters.

`location`
A structure specifying the profile location. With ColorSync 2.5, the location is always file-based, but that may not be true for future versions. Your code should always verify that the location structure contains a file specification before attempting to use it.

```
uniCodeNameCount
uniCodeName
asciiName
makeAndModel
digest
TPLDEV_TYPE_WPS_SPEED
deviceData
```

Discussion
The ColorSync Manager defines the `CMProfileIterateData` structure to provide your `CMProfileIterateProcPtr` (page 106) callback routine with a description of a profile during an iteration through the available profiles that takes place when you call `CMIterateColorSyncFolder` (page 57).

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

Declared In
`CMApplication.h`

CMProfileIterateUPP

Defines a universal procedure pointer to a profile iteration callback.

```
typedef CMProfileIterateProcPtr CMProfileIterateUPP;
```

Discussion
For more information, see the description of the `CMProfileIterateProcPtr` (page 106)) callback function.

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

Declared In
`CMApplication.h`

CMProfileLocation

Contains profile location information.

```
struct CMProfileLocation {
    short locType;
    CMProfLoc u;
};
typedef struct CMProfileLocation CMProfileLocation;
```

Fields
`locType`
The type of data structure that the `u` field's `CMProfLoc` union holds—a file specification, a handle, a pointer, or a universal procedure pointer. To specify the type, you use the constants defined in the enumeration described in "Profile Location Type" (page 244).

`u`
A union of type `CMProfLoc` (page 169) identifying the profile location.

Discussion
Your application passes a profile location structure of type `CMProfileLocation` when it calls:

- the function CMOpenProfile (page 63), specifying the location of a profile to open
- the CMNewProfile (page 62), CWNewLinkProfile (page 310), or CMCopyProfile (page 28) functions, specifying the location of a profile to create or duplicate

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

Declared In
CMApplication.h

CMProfileMD5

Defines a data type for an MD5 digest.

```
typedef  unsigned char CMProfileMD5[16];
```

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

Declared In
CMICCProfile.h

CMProfileName

Contains profile name and length.

```
struct CMProfileName {
    UniCharCount profileNameLength;
    UniChar profileName[256];
};
typedef struct CMProfileName CMProfileName;
```

CMProfileNamePtr

Defines a poitner to a profile name data structure.

```
typedef CMProfileName* CMProfileNamePtr;
```

CMProfileRef

Defines a reference to an opaque data type that specifies profile information.

```
typedef struct OpaqueCMProfileRef * CMProfileRef;
```

Discussion
A profile reference is the means by which your application gains access to a profile. Several ColorSync Manager functions return a profile reference to your application. Your application then passes it as a parameter on subsequent calls to other ColorSync Manager functions that use profiles.

The ColorSync Manager returns a unique profile reference in response to each individual call to the `CMOpenProfile` (page 63), `CMCopyProfile` (page 28) , and `CMNewProfile` (page 62) functions. This allows multiple applications concurrent access to a profile. The ColorSync Manager defines an abstract private data structure of type `OpaqueCMProfileRef` for the profile reference.

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

Declared In
`CMTypes.h`

CMProfileResponse

```
struct CMProfileResponse {
    UInt16 counts[9];
    UInt16 data[1];
};
typedef struct CMProfileResponse CMProfileResponse;
```

Fields
`counts`
`data`

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

Declared In
`CMICCProfile.h`

CMProfileSearchRecord

```
struct CMProfileSearchRecord {
    CMHeader header;
    UInt32 fieldMask;
    UInt32 reserved[2];
};
typedef struct CMProfileSearchRecord CMProfileSearchRecord;
typedef CMProfileSearchRecord * CMProfileSearchRecordPtr;
```

Fields
`header`
`fieldMask`
`reserved`

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

Declared In
`CMApplication.h`

CMProfileSearchRef

Defines a reference to an opaque profile search object.

```
typedef struct OpaqueCMProfileSearchRef * CMProfileSearchRef;
```

Discussion
A search result consists of a list of profiles matching certain search criteria. When your application calls the function `CMNewProfileSearch` (page 296) to search in the ColorSync Profiles folder for profiles that meet certain criteria, the ColorSync Manager returns a reference to an internal private data structure containing the search result. Your application passes the search result reference to these ColorSync functions:

- `CMUpdateProfileSearch` (page 308) updates a search result list.
- `CMDisposeProfileSearch` (page 285) disposes of a search result list.
- `CMSearchGetIndProfile` (page 302) opens a reference to a profile at a specific position in a search result list.
- `CMSearchGetIndProfileFileSpec` (page 302) obtains the file specification for a profile in a search result list.

The ColorSync Manager uses an abstract private data structure of type `OpaqueCMProfileSearchRef` in defining the search result reference.

Version Notes
This type is not recommended for use in ColorSync 2.5.

This type does not take advantage of the profile cache added in ColorSync version 2.5. It is used with the searching described in "Searching for Profiles Prior to ColorSync 2.5". See `CMProfileIterateData` (page 164) for information on data structures used with searching in version 2.5.

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

Declared In
`CMTypes.h`

CMProfileSequenceDescType

```
struct CMProfileSequenceDescType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 count;
    char data[1];
};
typedef struct CMProfileSequenceDescType CMProfileSequenceDescType;
```

Fields

`typeDescriptor`

`reserved`

`count`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMProfLoc

Defines a uniont that identifies the location of a profile.

```
union CMProfLoc {
    CMFileLocation fileLoc;
    CMHandleLocation handleLoc;
    CMPtrLocation ptrLoc;
    CMProcedureLocation procLoc;
    CMPathLocation pathLoc;
    CMBufferLocation bufferLoc;
};
typedef union CMProfLoc CMProfLoc;
```

Fields

`fileLoc`

A data structure containing a file system specification record specifying the location of a profile disk file.

`handleLoc`

A data structure containing a handle that indicates the location of a profile in relocatable memory.

`ptrLoc`

A data structure containing a pointer that points to a profile in nonrelocatable memory.

`procLoc`

A data structure containing a universal procedure pointer that points to a profile access procedure supplied by you. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed.

`pathLoc`
`bufferLoc`

Discussion
You use a union of type `CMProfLoc` to identify the location of a profile. You specify the union in the `u` field of the data type `CMProfileLocation` (page 165). Your application passes a pointer to a `CMProfileLocation` structure when it calls the `CMOpenProfile` (page 63) function to identify the location of a profile or the `CMNewProfile` (page 62), `CMCopyProfile` (page 28) , or `CWNewLinkProfile` (page 310) functions to specify the location for a newly created profile.

You also pass a pointer to a `CMProfileLocation` structure to the `NCMGetProfileLocation` (page 88) and `CMGetProfileLocation` (page 293) functions to get the location of an existing profile. The `NCMGetProfileLocation` function is available starting with ColorSync version 2.5. It differs from its predecessor, `CMGetProfileLocation`, in that the newer version has a parameter for the size of the location structure for the specified profile.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`CMApplication.h`

CMPS2CRDVMSizeType

Defines the Apple-defined `'psvm'` optional tag.

```
struct CMPS2CRDVMSizeType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 count;
    CMIntentCRDVMSize intentCRD[1];
};
typedef struct CMPS2CRDVMSizeType CMPS2CRDVMSizeType;
```

Fields
`typeDescriptor`
The `'psvm'` tag signature.
`reserved`
Reserved for future use.
`count`
The number of entries in the `intentCRD` array. You should specify at least four entries: 0, 1, 2, and 3.
`intentCRD`
A variable-sized array of four or more members defined by the `CMIntentCRDSize` data type.

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

Declared In
`CMICCProfile.h`

CMPtrLocation

Contains a pointer specification for a profile stored in nonrelocatable memory.

```
struct CMPtrLocation {
    Ptr p;
};
typedef struct CMPtrLocation CMPtrLocation;
```

Fields

`p`

A data structure of type `Ptr` holding a pointer that points to the location of a profile in memory.

Discussion

Your application uses the `CMPtrLocation` structure to provide a pointer specification for a profile stored in nonrelocatable memory. You provide the pointer specification structure in the `CMProfileLocation` structure's `u` field to point to an existing profile.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMRGBColor

Contains a color value expressed in the RGB color space.

```
struct CMRGBColor {
    UInt16 red;
    UInt16 green;
    UInt16 blue;
};
typedef struct CMRGBColor CMRGBColor;
```

Fields

`red`

`green`

`blue`

Discussion

A color value expressed in the RGB color space is composed of `red`, `green`, and `blue` component values. Each color component is expressed as a numeric value within the range of 0 to 65535.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMS15Fixed16ArrayType

```
struct CMS15Fixed16ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    Fixed value[1];
};
typedef struct CMS15Fixed16ArrayType CMS15Fixed16ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMScreeningChannelRec

```
struct CMScreeningChannelRec {
    Fixed frequency;
    Fixed angle;
    UInt32 spotFunction;
};
typedef struct CMScreeningChannelRec CMScreeningChannelRec;
```

Fields

`frequency`

`angle`

`spotFunction`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMScreeningType

```
struct CMScreeningType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 screeningFlag;
    UInt32 channelCount;
    CMScreeningChannelRec channelInfo[1];
};
typedef struct CMScreeningType CMScreeningType;
```

Fields

`typeDescriptor`

`reserved`

`screeningFlag`

`channelCount`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMSearchRecord

Contains information needed for a search.

```
struct CMSearchRecord {
    OSType CMMType;
    OSType profileClass;
    OSType dataColorSpace;
    OSType profileConnectionSpace;
    UInt32 deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 profileFlags;
    UInt32 searchMask;
    CMProfileFilterUPP filter;
};
typedef struct CMSearchRecord CMSearchRecord;
```

Fields

`CMMType`

The signature of a CMM. The signature of the default CMM is specified by the `kDefaultCMMSignature` constant.

`profileClass`

The class signature identifying the type of profile to search for. For a list of profile class signatures, see "Profile Classes" (page 240).

`dataColorSpace`

A data color space. For a list of the color space signatures, see "Color Space Signatures" (page 210).

`profileConnectionSpace`

A profile connection color space. The signatures for the two profile connection spaces supported by ColorSync, `cmXYZData` and `cmLabData`, are described in "Color Space Signatures" (page 210).

`deviceManufacturer`
: The signature of the manufacturer.

`deviceModel`
: The model of a device.

`deviceAttributes`
: Attributes for a particular device setup, such as media, paper, and ink types.

`profileFlags`
: Flags that indicate hints for the preferred CMM, such as quality, speed, and memory options. In most cases, you will not want to search for profiles based on the flags settings.

`searchMask`
: A bitmask that specifies the search record fields to use in the profile search.

`filter`
: A pointer to an application-supplied function that determines whether to exclude a profile from the profile search result list. For more information, see the function `CMProfileFilterProcPtr` (page 105).

Discussion
Your application supplies a search record of type `CMSearchRecord` as the `searchSpec` parameter to the function `CMNewProfileSearch` (page 296). The search record structure provides the ColorSync Manager with search criteria to use in determining which version 2.x profiles to include in the result list and which to filter out.

Most of the fields in the `CMSearchRecord` structure are identical to corresponding fields in the `CM2Header` structure for version 2.x profiles. When you set a bit in the `searchMask` field of the `CMSearchRecord` structure, you cause the search criteria to include the data specified by that bit. For example, if you set the `cmMatchProfileCMMType` bit, the search result will not include a profile unless the data in the profile header's `CMMType` field matches the data you specify in the `CMSearchRecord` structure's `CMMType` field.

If you specify a bit in the `searchMask` field, you must supply information in the `CMSearchRecord` field that corresponds to that bit.

The ColorSync Manager preserves the search criteria internally along with the search result list until your application calls the `CMDisposeProfileSearch` function to release the memory. This allows your application to call the `CMUpdateProfileSearch` function to update the search result if the ColorSync Profiles folder contents change without needing to provide the search specification again.

Version Notes
This type is not recommended for use in ColorSync 2.5.

You cannot use the ColorSync Manager search functions to search for ColorSync 1.0 profiles.

This type does not take advantage of the profile cache added in ColorSync version 2.5. It is used with the searching described in "Searching for Profiles Prior to ColorSync 2.5". See `CMProfileIterateData` (page 164) for information on data structures used with searching in version 2.5.

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

Declared In
`CMApplication.h`

CMSignatureType

```
struct CMSignatureType {
    OSType typeDescriptor;
    UInt32 reserved;
    OSType signature;
};
typedef struct CMSignatureType CMSignatureType;
```

Fields

`typeDescriptor`

`reserved`

`signature`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMTagElemTable

```
struct CMTagElemTable {
    UInt32 count;
    CMTagRecord tagList[1];
};
typedef struct CMTagElemTable CMTagElemTable;
```

Fields

`count`

`tagList`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMTagRecord

```
struct CMTagRecord {
    OSType tag;
    UInt32 elementOffset;
    UInt32 elementSize;
};
typedef struct CMTagRecord CMTagRecord;
```

Fields

`tag`

`elementOffset`

`elementSize`

Availability

Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMTextDescriptionType

```
struct CMTextDescriptionType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 ASCIICount;
    UInt8 ASCIIName[2];
};
typedef struct CMTextDescriptionType CMTextDescriptionType;
```

Fields
typeDescriptor
reserved
ASCIICount
ASCIIName
UniCodeCode
UniCodeCount
UniCodeName
ScriptCodeCode
ScriptCodeCount
ScriptCodeName

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

Declared In
CMICCProfile.h

CMTextType

```
struct CMTextType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 text[1];
};
typedef struct CMTextType CMTextType;
```

Fields
typeDescriptor
reserved
text

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

Declared In
CMICCProfile.h

CMU16Fixed16ArrayType

```
struct CMU16Fixed16ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 value[1];
};
typedef struct CMU16Fixed16ArrayType CMU16Fixed16ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUcrBgType

```
struct CMUcrBgType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 ucrCount;
    UInt16 ucrValues[1];
};
typedef struct CMUcrBgType CMUcrBgType;
```

Fields

`typeDescriptor`

`reserved`

`ucrCount`

`ucrValues`

`bgCount`

`bgValues`

`ucrbgASCII`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt16ArrayType

```
struct CMUInt16ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt16 value[1];
};
typedef struct CMUInt16ArrayType CMUInt16ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt32ArrayType

```
struct CMUInt32ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 value[1];
};
typedef struct CMUInt32ArrayType CMUInt32ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt64ArrayType

```
struct CMUInt64ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 value[1];
};
typedef struct CMUInt64ArrayType CMUInt64ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt8ArrayType

```
struct CMUInt8ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 value[1];
};
typedef struct CMUInt8ArrayType CMUInt8ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUnicodeTextType

```
struct CMUnicodeTextType {
    OSType typeDescriptor;
    UInt32 reserved;
    UniChar text[1];
};
typedef struct CMUnicodeTextType CMUnicodeTextType;
```

Fields

`typeDescriptor`

`reserved`

`text`

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

Declared In
`CMICCProfile.h`

CMVideoCardGamma

Contains video gamma data to store with a video gamma profile tag.

```
struct CMVideoCardGamma {
    UInt32 tagType
    union {
        CMVideoCardGammaTable table;
        CMVideoCardGammaFormula formula;
    } u;
};
typedef struct CMVideoCardGamma CMVideoCardGamma;
```

Fields

`tagType`

A "Video Card Gamma Storage Types" (page 259) constant that specifies the format of the data currently stored in the union. To determine the type of structure present in a specific instance of the `CMVideoCardGamma` structure, you test this union tag. If you are setting up a `CMVideoCardGamma` structure to store video card gamma data, you set `tagType` to a constant value that identifies the structure type you are using. The possible constant values are described in "Video Card Gamma Storage Types" (page 259).

`table`

A structure of type `CMVideoCardGammaTable`. If the `tagType` field has the value `cmVideoCardGammaTableType`, the `CMVideoCardGamma` structure's union field should be treated as a table, as described in `CMVideoCardGammaTable` (page 182).

`formula`

Discussion
The ColorSync Manager defines the `CMVideoCardGamma` data structure to specify the video gamma data to store with a video gamma profile tag. The structure is a union that can store data in either table or formula format.

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

Declared In
CMICCProfile.h

CMVideoCardGammaFormula

```
struct CMVideoCardGammaFormula {
    Fixed redGamma;
    Fixed redMin;
    Fixed redMax;
    Fixed greenGamma;
    Fixed greenMin;
    Fixed greenMax;
    Fixed blueGamma;
    Fixed blueMin;
    Fixed blueMax;
};
typedef struct CMVideoCardGammaFormula CMVideoCardGammaFormula;
```

Fields

`redGamma`
: The gamma value for red. It must be greater than 0.0.

`redMin`
: The minimum gamma value for red. It must be greater than 0.0 and less than 1.0.

`redMax`
: The maximum gamma value for red. It must be greater than 0.0 and less than 1.0.

`greenGamma`
: The gamma value for green. It must be greater than 0.0.

`greenMin`
: The minimum gamma value for green. It must be greater than 0.0 and less than 1.0.

`greenMax`
: The maximum gamma value for green. It must be greater than 0.0 and less than 1.0.

`blueGamma`
: The gamma value for blue. It must be greater than 0.0.

`blueMin`
: The minimum gamma value for blue. It must be greater than 0.0 and less than 1.0.

`blueMax`
: The maximum gamma value for blue. It must be greater than 0.0 and less than 1.0.

Discussion
The ColorSync Manager defines the `CMVideoCardGammaFormula` data structure to specify video card gamma data by providing three values each for red, blue and green gamma. The values represent the actual gamma, the minimum gamma, and the maximum gamma for each color. Specifying video gamma information by formula takes less space than specifying it with a table, but the results may be less precise.

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

Declared In
CMICCProfile.h

CMVideoCardGammaTable

```
struct CMVideoCardGammaTable {
    UInt16 channels;
    UInt16 entryCount;
    UInt16 entrySize;
    char data[1];
};
typedef struct CMVideoCardGammaTable CMVideoCardGammaTable;
```

Fields

`channels`

Number of gamma channels (1 or 3). If `channels` is set to 1 then the red, green, and blue lookup tables (LUTs) of the video card will be loaded with the same data. If `channels` is set to 3, then if the video card supports separate red, green, and blue LUTs, then the video card LUTs will be loaded with the data for the three channels from the `data` array.

`entryCount`

Number of entries per channel (1-based). The number of entries must be greater than or equal to 2.

`entrySize`

Size in bytes of each entry.

`data`

Variable-sized array of data. The size of the data is equal to `channels*entryCount*entrySize`.

Discussion

The ColorSync Manager defines the `CMVideoCardGammaTable` data structure to specify video card gamma data in table format. You specify the number of channels, the number of entries per channel, and the size of each entry. The last field in the structure is an array of size one that serves as the start of the table data. The actual size of the array is equal to the number of channels times the number of entries times the size of each entry.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMVideoCardGammaType

Specifies a video card gamma profile tag.

```
struct CMVideoCardGammaType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMVideoCardGamma gamma;
};
typedef struct CMVideoCardGammaType CMVideoCardGammaType;
```

Fields

`typeDescriptor`

The signature type for a video card gamma tag. There is currently only one type possible, `cmSigVideoCardGammaType`.

`reserved`

This field is reserved and must contain the value 0.

`gamma`

A structure that specifies the video card gamma data for the profile tag, as described in `CMVideoCardGamma` (page 180).

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

Declared In
`CMICCProfile.h`

CMViewingConditionsType

```
struct CMViewingConditionsType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMFixedXYZColor illuminant;
    CMFixedXYZColor surround;
    UInt32 stdIlluminant;
};
typedef struct CMViewingConditionsType CMViewingConditionsType;
```

Fields
`typeDescriptor`
`reserved`
`illuminant`
`surround`
`stdIlluminant`

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

Declared In
`CMICCProfile.h`

CMWorldRef

Defines an opaque data type used for color-matching and color-checking sessions.

```
typedef struct OpaqueCMWorldRef * CMWorldRef;
```

Discussion
Your application passes a color world reference as a parameter on calls to functions to perform color-matching and color-checking sessions and to dispose of the color world. When your application calls the function `NCWNewColorWorld` (page 90) and the function `CWConcatColorWorld` (page 83) to allocate a color world for color-matching and color-checking sessions, the ColorSync Manager returns a reference to the color world. The ColorSync Manager defines an abstract private data structure of type `OpaqueCMWorldRef` for the color world reference.

The color world is affected by the rendering intent, lookup flag, gamut flag, and quality flag of the profiles that make up the color world. For more information, see "Rendering Intent Values for Version 2.x Profiles" (page 253), "Flag Mask Definitions for Version 2.x Profiles" (page 224), and "Quality Flag Values for Version 2.x Profiles" (page 252).

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

Declared In
CMTypes.h

CMXYZColor

Contains values for a color specified in XYZ color space.

```
struct CMXYZColor {
    CMXYZComponent X;
    CMXYZComponent Y;
    CMXYZComponent Z;
};
typedef struct CMXYZColor CMXYZColor;
typedef CMXYZColor XYZColor;
```

Fields
X
Y
Z

Discussion
Three color component values defined by the `CMXYZComponent` type definition combine to form a color value specified in the XYZ color space. The color value is defined by the `CMXYZColor` type definition.

Your application uses the `CMXYZColor` data structure to specify a color value in the `CMColor` union to use in general purpose color matching, color checking, or color conversion. You also use the `CMXYZColor` data structure to specify the XYZ white point reference used in the conversion of colors to or from the XYZ color space.

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

Declared In
CMICCProfile.h

CMXYZComponent

```
typedef UInt16 CMXYZComponent;
```

Discussion
Three components combine to express a color value defined by the `CMXYZColor` type definition in the XYZ color space. Each color component is described by a numeric value defined by the `CMXYZComponent` type definition. A component value of type `CMXYZComponent` is expressed as a 16-bit value. This is formatted as an unsigned value with 1 bit of integer portion and 15 bits of fractional portion.

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

Declared In
CMICCProfile.h

CMXYZType

```
struct CMXYZType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMFixedXYZColor XYZ[1];
};
typedef struct CMXYZType CMXYZType;
```

Fields

`typeDescriptor`

`reserved`

`XYZ`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMYKColor

```
typedef CMCMYKColor CMYKColor;
```

CMYxyColor

Contains values for a color expresed in the Yxy color space.

```
struct CMYxyColor {
    UInt16 capY;
    UInt16 x;
    UInt16 y;
};
typedef struct CMYxyColor CMYxyColor;
```

Fields

`capY`

`x`

`y`

Discussion

A color value expressed in the Yxy color space is composed of `capY`, `x`, and `y` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 which maps to 0 to 1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

NCMConcatProfileSet

```
struct NCMConcatProfileSet {
    OSType cmm;
    UInt32 flags;
    UInt32 flagsMask;
    UInt32 profileCount;
    NCMConcatProfileSpec profileSpecs[1];
};
typedef struct NCMConcatProfileSet NCMConcatProfileSet;
```

Fields

`cmm`

`flags`

`flagsMask`

`profileCount`

`profileSpecs`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

NCMConcatProfileSpec

```
struct NCMConcatProfileSpec {
    UInt32 renderingIntent;
    UInt32 transformTag;
    CMProfileRef profile;
};
typedef struct NCMConcatProfileSpec NCMConcatProfileSpec;
```

Fields

`renderingIntent`

`transformTag`

`profile`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

NCMDeviceProfileInfo

```
struct NCMDeviceProfileInfo {
    UInt32 dataVersion;
    CMDeviceProfileID profileID;
    CMProfileLocation profileLoc;
    CFDictionaryRef profileName;
    CMDeviceProfileScope profileScope;
    UInt32 reserved;
};
typedef struct NCMDeviceProfileInfo NCMDeviceProfileInfo;
```

Fields

`dataVersion`

`profileID`

`profileLoc`

`profileName`

`profileScope`

`reserved`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMDeviceIntegration.h`

Constants

Abstract Color Space Constants

Specify values that represent general color spaces.

```
enum {
    cmNoSpace = 0x0000,
    cmRGBSpace = 0x0001,
    cmCMYKSpace = 0x0002,
    cmHSVSpace = 0x0003,
    cmHLSSpace = 0x0004,
    cmYXYSpace = 0x0005,
    cmXYZSpace = 0x0006,
    cmLUVSpace = 0x0007,
    cmLABSpace = 0x0008,
    cmReservedSpace1 = 0x0009,
    cmGraySpace = 0x000A,
    cmReservedSpace2 = 0x000B,
    cmGamutResultSpace = 0x000C,
    cmNamedIndexedSpace = 0x0010,
    cmMCFiveSpace = 0x0011,
    cmMCSixSpace = 0x0012,
    cmMCSevenSpace = 0x0013,
    cmMCEightSpace = 0x0014,
    cmAlphaPmulSpace = 0x0040,
    cmAlphaSpace = 0x0080,
    cmRGBASpace = cmRGBSpace + cmAlphaSpace,
    cmGrayASpace = cmGraySpace + cmAlphaSpace,
    cmRGBAPmulSpace = cmRGBASpace + cmAlphaPmulSpace,
    cmGrayAPmulSpace = cmGrayASpace + cmAlphaPmulSpace
};
```

Constants

`cmNoSpace`

The ColorSync Manager does not use this constant.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBSpace`

An RGB color space composed of red, green, and blue components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmCMYKSpace`

A CMYK color space composed of cyan, magenta, yellow, and black. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmHSVSpace`

An HSV color space composed of hue, saturation, and value components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmHLSSpace`
: An HLS color space composed of hue, lightness, and saturation components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmYXYSpace`
: A Yxy color space composed of Y, x, and y components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZSpace`
: An XYZ color space composed of X, Y, and Z components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLUVSpace`
: An L*u*v* color space composed of L*, u*, and v* components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLABSpace`
: An L*a*b* color space composed of L*, a*, b* components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmReservedSpace1`
: This field is reserved for use by QuickDraw GX.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGraySpace`
: A luminance color space with a single component, gray.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmReservedSpace2`
: This field is reserved for use by QuickDraw GX.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGamutResultSpace`
A color space for the resulting bitmap pointed to by the `resultBitMap` field of the function `CWMatchColors` (page 87). A bitmap never uses this constant alone. Instead, it uses the constant `cmGamutResult1Space`, which combines `cmGamutResultSpace` and `cmOneBitDirectPacking` to define a bitmap that is 1 bit deep.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmNamedIndexedSpace`
A named indexed color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCFiveSpace`
A five-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSixSpace`
A six-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSevenSpace`
A seven-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCEightSpace`
An eight-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaPmulSpace`
A premultiplied alpha channel component is added to the color value.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaSpace`
An alpha channel component is added to the color value.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBASpace`
An RGB color space composed of red, green, and blue color value components and an alpha channel component. ColorSync does not currently support bitmaps that use this constant alone. Instead, this constant indicates the presence of an alpha channel in combination with `cmLong8ColorPacking` to indicate 8-bit packing format and `cmAlphaFirstPacking` to indicate the position of the alpha channel as the first component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGrayASpace`
A luminance color space with two components, a gray component followed by an alpha channel component. Each component value is 16 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

Discussion
The data type `CMBitmap` (page 123) defines a bitmap for an image whose colors can be matched with the function `CWMatchColors` (page 87) or color-checked with the function `CWCheckColors` (page 81).

The `space` field of the `CMBitmap` type definition identifies the color space in which the colors of the bitmap image are specified. A color space is characterized by a number of components or dimensions, with each component carrying a numeric value. These values together make up the color value. A color space also specifies the format in which the color value is stored. For bitmaps in which color values are packed, the `space` field of the `CMBitmap` data type holds a constant that defines the color space and the packing format.

For the `CWMatchBitmap` function to perform color matching successfully, the color space specified in the `CMBitmap` data type's `space` field must correspond to the color space specified in the profile's `dataColorSpace` field. The source bitmap and source profile values must match and the destination bitmap and destination profile values must match. For the `CWCheckBitMap` function to perform color checking successfully, the source profile's `dataColorSpace` field value and the `space` field value of the source bitmap must specify the same color space. These functions will execute successfully as long as the color spaces are the same without regard for the packing format specified by the bitmap.

This enumeration defines constants for abstract color spaces which, when combined with a packing format constant as described in "Color Packing for Color Spaces" (page 198), can be used in the `space` field of the `CMBitmap` structure. The combined constants are shown in "Color Space Constants With Packing Formats" (page 203).

Version Notes
The constants `cmRGBASpace` and `cmGrayASpace` were moved to this enum from "Color Space Constants With Packing Formats" (page 203) in ColorSync version 2.5.

Calibrator Name Prefix

Specify an interface for new ColorSync monitor calibrators (ColorSync 2.6 and greater)

```
enum {
    kCalibratorNamePrefix = 'cali'
};
```

Constants
`kCalibratorNamePrefix`
Available in Mac OS X v10.0 and later.

Declared in `CMCalibrator.h`.

Channel Encoding Format

Specify an encoding format for sRGB64.

```
enum {
    cmSRGB16ChannelEncoding = 0x00010000
};
```

Constants

`cmSRGB16ChannelEncoding`
Used for sRGB64 encoding (±3.12 format)
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Chromatic Adaptation Values

Specify a transformation to use for chromaric adaptation.

```
typedef UInt32 CMChromaticAdaptation;
enum {
    cmUseDefaultChromaticAdaptation = 0,
    cmLinearChromaticAdaptation = 1,
    cmVonKriesChromaticAdaptation = 2,
    cmBradfordChromaticAdaptation = 3
};
```

Constants

`cmUseDefaultChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

`cmLinearChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

`cmVonKriesChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

`cmBradfordChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

CMM Function Selectors

Define selectors used for component-based CMM functions.

```
enum {
    kCMMOpen = -1,
    kCMMClose = -2,
    kCMMGetInfo = -4,
    kNCMMInit = 6,
    kCMMMatchColors = 1,
    kCMMCheckColors = 2,
    kCMMValidateProfile = 8,
    kCMMMatchBitmap = 9,
    kCMMCheckBitmap = 10,
    kCMMConcatenateProfiles = 5,
    kCMMConcatInit = 7,
    kCMMNewLinkProfile = 16,
    kNCMMConcatInit = 18,
    kNCMMNewLinkProfile = 19,
    kCMMGetPS2ColorSpace = 11,
    kCMMGetPS2ColorRenderingIntent = 12,
    kCMMGetPS2ColorRendering = 13,
    kCMMGetPS2ColorRenderingVMSize = 17,
    kCMMFlattenProfile = 14,
    kCMMUnflattenProfile = 15,
    kCMMInit = 0,
    kCMMGetNamedColorInfo = 70,
    kCMMGetNamedColorValue = 71,
    kCMMGetIndNamedColorValue = 72,
    kCMMGetNamedColorIndex = 73,
    kCMMGetNamedColorName = 74,
    kCMMMatchPixMap = 3,
    kCMMCheckPixMap = 4
};
```

Constants

`kCMMOpen`

Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMClose`

Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetInfo`

Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kNCMMInit`

In response to this request code, your CMM should initialize any private data it will need for the color session and for subsequent requests from the calling application or driver. Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMMatchColors`

In response to this request code, your CMM should match the colors in the `myColors` parameter to the color gamut of the destination profile and replace the color-list color values with the matched colors. Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMCheckColors`

In response to this request code, your CMM should test the given list of colors in the `myColors` parameter against the gamut specified by the destination profile and report if the colors fall within a destination device's color gamut. For more information, see the function `CWCheckColors` (page 81). Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMValidateProfile`

In response to this request code, your CMM should test the profile whose reference is passed in the `prof` parameter to determine if the profile contains the minimum set of elements required for a profile of its type. For more information, see the function `CMValidateProfile` (page 79).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMMatchBitmap`

In response to this request code, your CMM must match the colors of the source image bitmap pointed to by the `bitmap` parameter to the gamut of the destination device using the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWMatchBitmap` (page 86).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMCheckBitmap`

In response to this request code, your CMM must check the colors of the source image bitmap pointed to by the `bitmap` parameter against the gamut of the destination device using the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWCheckBitmap` (page 80).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMConcatenateProfiles`

This request code is for backward compatibility with ColorSync 1.0.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMConcatInit`

In response to this request code, your CMM should initialize any private data your CMM will need for a color session involving the set of profiles specified by the profile array pointed to by the `profileSet` parameter. Your function should also initialize any additional private data needed in handling subsequent calls pertaining to this component instance. For more information, see the function `CWConcatColorWorld` (page 83).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMNewLinkProfile`

In response to this request code, your CMM must create a single device link profile of type `DeviceLink` that includes the profiles passed to you in the array pointed to by the `profileSet` parameter. For more information, see the function `CWNewLinkProfile` (page 310).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kNCMMConcatInit`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kNCMMNewLinkProfile`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorSpace`

In response to this request code, your CMM must obtain or derive the color space element data from the source profile whose reference is passed to your function in the `srcProf` parameter and pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see the function `CMGetPS2ColorSpace` (page 53).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorRenderingIntent`

In response to this request code, your CMM must obtain the color-rendering intent from the header of the source profile whose reference is passed to your function in the `srcProf` parameter and then pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see the function `CMGetPS2ColorRenderingIntent` (page 51).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorRendering`

In response to this request code, your CMM must obtain the rendering intent from the source profile's header and generate the color rendering dictionary (CRD) data from the destination profile, and then pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see the function `CMGetPS2ColorRendering` (page 50).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorRenderingVMSize`

In response to this request code, your CMM must obtain or assess the maximum virtual memory (VM) size of the color rendering dictionary (CRD) specified by the destination profile. You must return the size of the CRD for the rendering intent specified by the source profile. For more information, see the function `CMGetPS2ColorRenderingVMSize` (page 52).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMFlattenProfile`

In response to this request code, your CMM must extract the profile data from the profile to flatten, identified by the `prof` parameter, and pass the profile data to the function specified in the `proc` parameter. For more information, see the function `CMFlattenProfile` (page 286).

Changed in ColorSync 2.5: Starting with ColorSync version 2.5, the ColorSync Manager calls the function provided by the calling program directly, without going through the preferred, or any, CMM. Your CMM only needs to handle this request code for versions of ColorSync prior to version 2.5.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMUnflattenProfile`

In response to this request code, your CMM must create a temporary file in which to store the profile data you receive from the low-level data-transfer function supplied by the calling application or driver. Your function must return the file specification.

Changed in ColorSync 2.5: Starting with ColorSync version 2.5, the ColorSync Manager calls the function provided by the calling program directly, without going through the preferred, or any, CMM. Your CMM only needs to handle this request code for versions of ColorSync prior to version 2.5.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMInit`

This request code is provided for backward compatibility with ColorSync 1.0. A CMM that supports ColorSync 1.0 profiles should respond to this request code by initializing any private data required for the color-matching or gamut-checking session to be held as indicated by subsequent request codes. If your CMM supports only ColorSync 1.0 profiles or both ColorSync 1.0 profiles and ColorSync Manager version 2.x profiles, you must support this request code. If you support only ColorSync Manager version 2.x profiles, you should return an unimplemented error in response to this request code.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorInfo`

In response to this request code, your CMM extracts named color data from the profile whose reference is passed in the `srcProf` parameter. For more information, see the function `CMGetNamedColorInfo` (page 41).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorValue`

In response to this request code, your CMM extracts device and profile connection space (PCS) color values for a specific color name from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetNamedColorValue` (page 43).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetIndNamedColorValue`

In response to this request code, your CMM extracts device and PCS color values for a specific named color index from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetIndNamedColorValue` (page 38).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorIndex`

In response to this request code, your CMM extracts a named color index for a specific color name from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetNamedColorIndex` (page 40).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorName`

In response to this request code, your CMM extracts a named color name for a specific named color index from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetNamedColorName` (page 42).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMMatchPixMap`

In response to this request code, your CMM must match the colors of the pixel map image pointed to by the `myPixMap` parameter to the gamut of the destination device, replacing the original pixel colors with their corresponding colors as specified in the data color space of the destination device's color gamut. To perform the matching, you use the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWMatchPixMap` (page 268).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMCheckPixMap`

In response to this request code, your CMM must check the colors of the pixel map image pointed to by the `myPixMap` parameter against the gamut of the destination device to determine if the pixel colors are within the gamut of the destination device and report the results. To perform the check, you use the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWCheckPixMap` (page 266).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

Discussion

Your CMM must respond to the ColorSync Manager required request codes. When a CMM receives a required request code from the ColorSync Manager, the CMM must determine the nature of the request, perform the appropriate processing, set an error code if necessary, and return an appropriate function result to the Component Manager. The required request codes are:

- `kNCMMInit`
- `kCMMMatchColors`
- `kCMMCheckColors`
- `kCMMInit`

Your CMM should respond to the rest of the ColorSync Manager request codes defined by this enumeration, but it is not required to do so.

Color Management Module Component Interface

Specify a CMM interface version.

```
enum {
    CMMInterfaceVersion = 1
};
```

Constants

`CMMInterfaceVersion`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

Discussion

If your CMM supports the ColorSync Manager version 2.x, it should return the constant defined by the following enumeration when the Component Manager calls your CMM with the `kComponentVersionSelect` request code.

In response to the `kComponentVersionSelect` request code, a CMM should set its entry point function's result to the CMM version number. The high-order 16 bits represent the major version and the low-order 16 bits represent the minor version. The `CMMInterfaceVersion` constant represents the major version number.

A CMM that only supports ColorSync 1.0 returns 0 for the major version in response to the version request.

The `kComponentVersionSelect` request code is one of four required Component Manager requests your CMM must handle.

Color Packing for Color Spaces

Specify how color values are stored.

```
enum {
    cmNoColorPacking = 0x0000,
    cmWord5ColorPacking = 0x0500,
    cmWord565ColorPacking = 0x0600,
    cmLong8ColorPacking = 0x0800,
    cmLong10ColorPacking = 0x0A00,
    cmAlphaFirstPacking = 0x1000,
    cmOneBitDirectPacking = 0x0B00,
    cmAlphaLastPacking = 0x0000,
    cm8_8ColorPacking = 0x2800,
    cm16_8ColorPacking = 0x2000,
    cm24_8ColorPacking = 0x2100,
    cm32_8ColorPacking = cmLong8ColorPacking,
    cm40_8ColorPacking = 0x2200,
    cm48_8ColorPacking = 0x2300,
    cm56_8ColorPacking = 0x2400,
    cm64_8ColorPacking = 0x2500,
    cm32_16ColorPacking = 0x2600,
    cm48_16ColorPacking = 0x2900,
    cm64_16ColorPacking = 0x2A00,
    cm32_32ColorPacking = 0x2700,
    cmLittleEndianPacking = 0x4000,
    cmReverseChannelPacking = 0x8000
};
```

Constants

`cmNoColorPacking`

This constant is not used for ColorSync bitmaps.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmWord5ColorPacking`

The color values for three 5-bit color channels are stored consecutively in 16-bits, with the highest order bit unused.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmWord565ColorPacking`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLong8ColorPacking`

The color values for three or four 8-bit color channels are stored consecutively in a 32-bit long. For three channels, this constant is combined with either `cmAlphaFirstPacking` or `cmAlphaLastPacking` to indicate whether the unused eight bits are located at the beginning or end.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLong10ColorPacking`

The color values for three 10-bit color channels are stored consecutively in a 32-bit long, with the two highest order bits unused.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaFirstPacking`
An alpha channel is added to the color value as its first component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmOneBitDirectPacking`
One bit is used as the pixel format. This storage format is used by the resulting bitmap pointed to by the `resultBitMap` field of the function `CWMatchColors` (page 87); the bitmap must be only 1 bit deep.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

`cm8_8ColorPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm16_8ColorPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm24_8ColorPacking`
The color values for three 8-bit color channels are stored in consecutive bytes, for a total of 24 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm32_8ColorPacking`
The color values for four 8-bit color channels are stored in consecutive bytes, for a total of 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm40_8ColorPacking`
The color values for five 8-bit color channels are stored in consecutive bytes, for a total of 40 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm48_8ColorPacking`
The color values for six 8-bit color channels are stored in consecutive bytes, for a total of 48 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm56_8ColorPacking`
The color values for seven 8-bit color channels are stored in consecutive bytes, for a total of 56 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm64_8ColorPacking`
The color values for eight 8-bit color channels are stored in consecutive bytes, for a total of 64 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm32_16ColorPacking`
The color values for two 16-bit color channels are stored in a 32-bit word.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm48_16ColorPacking`
The color values for three 16-bit color channels are stored in 48 consecutive bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm64_16ColorPacking`
The color values for four 16-bit color channels are stored in 64 consecutive bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm32_32ColorPacking`
The color value for a 32-bit color channel is stored in a 32-bit word.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

Discussion
The ColorSync bitmap data type `CMBitmap` (page 123) includes a field that identifies the color space in which the color values of the bitmap image are expressed. This enumeration defines the types of packing for a color space's storage format. The enumeration also defines an alpha channel that can be added as a component of a color value to define the degree of opacity or transparency of a color. These constants are combined with the constants described in "Abstract Color Space Constants" (page 187) to create values that identify a bitmap's color space. Your application does not specify color packing constants directly, but rather uses the combined constants, which are described in "Color Space Constants With Packing Formats" (page 203).

Version Notes
The constants `cm48_16ColorPacking` and `cm64_16ColorPacking` were added in ColorSync version 2.5.

Color Responses

Specify responses for ColorSync 1.0 specifications.

```
enum {
    cmGrayResponse = 0,
    cmRedResponse = 1,
    cmGreenResponse = 2,
    cmBlueResponse = 3,
    cmCyanResponse = 4,
    cmMagentaResponse = 5,
    cmYellowResponse = 6,
    cmUcrResponse = 7,
    cmBgResponse = 8,
    cmOnePlusLastResponse = 9
};
```

Constants

`cmGrayResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmRedResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmGreenResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmBlueResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmCyanResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmMagentaResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmYellowResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmUcrResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmBgResponse`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmOnePlusLastResponse`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

Color Space Constants With Packing Formats

Specifies bitmap spaces with a wide range of data formats appropriate for multiple platforms.

```
enum {
    cmGray8Space = cmGraySpace + cm8_8ColorPacking,
    cmGray16Space = cmGraySpace,
    cmGray16LSpace = cmGraySpace + cmLittleEndianPacking,
    cmGrayA16Space = cmGrayASpace + cm16_8ColorPacking,
    cmGrayA32Space = cmGrayASpace,
    cmGrayA32LSpace = cmGrayASpace + cmLittleEndianPacking,
    cmGrayA16PmulSpace = cmGrayAPmulSpace + cm16_8ColorPacking,
    cmGrayA32PmulSpace = cmGrayAPmulSpace,
    cmGrayA32LPmulSpace = cmGrayAPmulSpace + cmLittleEndianPacking,
    cmRGB16Space = cmRGBSpace + cmWord5ColorPacking,
    cmRGB16LSpace = cmRGBSpace + cmWord5ColorPacking + cmLittleEndianPacking,
    cmRGB565Space = cmRGBSpace + cmWord565ColorPacking,
    cmRGB565LSpace = cmRGBSpace + cmWord565ColorPacking + cmLittleEndianPacking,
    cmRGB24Space = cmRGBSpace + cm24_8ColorPacking,
    cmRGB32Space = cmRGBSpace + cm32_8ColorPacking,
    cmRGB48Space = cmRGBSpace + cm48_16ColorPacking,
    cmRGB48LSpace = cmRGBSpace + cm48_16ColorPacking + cmLittleEndianPacking,
    cmARGB32Space = cmRGBASpace + cm32_8ColorPacking + cmAlphaFirstPacking,
    cmARGB64Space = cmRGBASpace + cm64_16ColorPacking + cmAlphaFirstPacking,
    cmARGB64LSpace = cmRGBASpace + cm64_16ColorPacking + cmAlphaFirstPacking
+ cmLittleEndianPacking,
    cmRGBA32Space = cmRGBASpace + cm32_8ColorPacking + cmAlphaLastPacking,
    cmRGBA64Space = cmRGBASpace + cm64_16ColorPacking + cmAlphaLastPacking,
    cmRGBA64LSpace = cmRGBASpace + cm64_16ColorPacking + cmAlphaLastPacking
+ cmLittleEndianPacking,
   cmARGB32PmulSpace = cmRGBAPmulSpace + cm32_8ColorPacking + cmAlphaFirstPacking,
   cmARGB64PmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaFirstPacking,
   cmARGB64LPmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaFirstPacking
+ cmLittleEndianPacking,
    cmRGBA32PmulSpace = cmRGBAPmulSpace + cm32_8ColorPacking + cmAlphaLastPacking,
   cmRGBA64PmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaLastPacking,
   cmRGBA64LPmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaLastPacking
+ cmLittleEndianPacking,
    cmCMYK32Space = cmCMYKSpace + cm32_8ColorPacking,
    cmCMYK64Space = cmCMYKSpace + cm64_16ColorPacking,
    cmCMYK64LSpace = cmCMYKSpace + cm64_16ColorPacking + cmLittleEndianPacking,
    cmHSV32Space = cmHSVSpace + cmLong10ColorPacking,
    cmHLS32Space = cmHLSSpace + cmLong10ColorPacking,
    cmYXY32Space = cmYXYSpace + cmLong10ColorPacking,
    cmXYZ24Space = cmXYZSpace + cm24_8ColorPacking,
    cmXYZ32Space = cmXYZSpace + cmLong10ColorPacking,
    cmXYZ48Space = cmXYZSpace + cm48_16ColorPacking,
    cmXYZ48LSpace = cmXYZSpace + cm48_16ColorPacking + cmLittleEndianPacking,
    cmLUV32Space = cmLUVSpace + cmLong10ColorPacking,
    cmLAB24Space = cmLABSpace + cm24_8ColorPacking,
    cmLAB32Space = cmLABSpace + cmLong10ColorPacking,
    cmLAB48Space = cmLABSpace + cm48_16ColorPacking,
    cmLAB48LSpace = cmLABSpace + cm48_16ColorPacking + cmLittleEndianPacking,
    cmGamutResult1Space = cmOneBitDirectPacking + cmGamutResultSpace,
    cmNamedIndexed32Space = cm32_32ColorPacking + cmNamedIndexedSpace,
    cmNamedIndexed32LSpace = cm32_32ColorPacking + cmNamedIndexedSpace
+ cmLittleEndianPacking,
    cmMCFive8Space = cm40_8ColorPacking + cmMCFiveSpace,
    cmMCSix8Space = cm48_8ColorPacking + cmMCSixSpace,
    cmMCSeven8Space = cm56_8ColorPacking + cmMCSevenSpace,
    cmMCEight8Space = cm64_8ColorPacking + cmMCEightSpace
};
```

```
typedef UInt32 CMBitmapColorSpace;
```

Constants

`cmGray8Space`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGray16Space`
A luminance color space with a single 16-bit component, gray.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGray16LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA16Space`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32Space`
A luminance color space with two components, a gray component followed by an alpha channel component. Each component value is 16 bits.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA16PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32LPmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB16Space`
An RGB color space composed of red, green, and blue components whose values are packed with 5 bits of storage per component. The storage size for a color value expressed in this color space is 16 bits, with the high-order bit not used.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB16LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB565Space`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB565LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB24Space`

An RGB color space composed of red, green, and blue components whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 24 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB32Space`

An RGB color space composed of red, green, and blue components whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with bits 24-31 not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB48Space`

An RGB color space composed of red, green, and blue components whose values are packed with 16 bits of storage per component. The storage size for a color value expressed in this color space is 48 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB48LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB32Space`

An RGB color space composed of red, green, and blue color value components preceded by an alpha channel component whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB64Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB64LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA32Space`

An RGB color space composed of red, green, and blue color value components, followed by an alpha channel component. Values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA64Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA64LSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB32PmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB64PmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB64LPmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA32PmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA64PmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA64LPmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmCMYK32Space`
A CMYK color space composed of cyan, magenta, yellow, and black components whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmCMYK64Space`
A CMYK color space composed of cyan, magenta, yellow, and black components whose values are packed with 16 bits of storage per component. The storage size for a color value expressed in this color space is 64 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmCMYK64LSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmHSV32Space`
An HSV color space composed of hue, saturation, and value components whose values are packed with 10 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmHLS32Space`

An HLS color space composed of hue, lightness, and saturation components whose values are packed with 10 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmYXY32Space`

A Yxy color space composed of Y, x, and y components whose values are packed with 10 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ24Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ32Space`

An XYZ color space composed of X, Y, and Z components whose values are packed with 10 bits per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ48Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ48LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLUV32Space`

An L*u*v* color space composed of L*, u*, and v* components whose values are packed with 10 bits per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB24Space`

An L*a*b* color space composed of L*, a*, and b* components whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 24 bits. The 8-bit unsigned a* and b* channels are interpreted numerically as ranging from -128.0 to approximately 128.0.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB32Space`
An L*a*b* color space composed of L*, a*, and b* components whose values are packed with 10 bits per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used. The 10-bit unsigned a* and b* channels are interpreted numerically as ranging from -128.0 to approximately 128.0.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB48Space`
An L*a*b* color space composed of L*, a*, and b* components whose values are packed with 16 bits per component. The storage size for a color value expressed in this color space is 48 bits. The 16-bit unsigned a* and b* channels are interpreted numerically as ranging from -128.0 to approximately 128.0.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB48LSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGamutResult1Space`
A gamut result color space for the resulting bitmap pointed to by the `resultBitMap` field of the function `CWMatchColors` (page 87), with 1-bit direct packing. A pixel in the returned bitmap with value 1 (displayed as black) indicates an out-of-gamut color, while a pixel value of 0 (white) indicates a color that is in gamut.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmNamedIndexed32Space`
A color space where each color is stored as a single 32-bit value, specifying an index into a named color space. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmNamedIndexed32LSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCFive8Space`
A five-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 40 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSix8Space`
A six-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 48 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSeven8Space`

A seven-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 56 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCEight8Space`

A eight-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 64 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion

This enumeration defines constants for color spaces which can specify color values for a bitmap image. As a rule, these constants include a packing format, defined in "Color Packing for Color Spaces" (page 198). You can use these constants to set the `space` field of the `CMBitmap` type definition identifies the color space in which the colors of the bitmap image are specified, as described in "Abstract Color Space Constants" (page 187).

Version Notes

The constants `cmRGBASpace` and `cmGrayASpace` were moved to "Abstract Color Space Constants" (page 187) in ColorSync version 2.5.

The constants `cmGray16Space`, `cmGrayA32Space`, `cmRGB48Space`, `cmCMYK64Space`, and `cmLAB48Space` were added in ColorSync version 2.5.

Color Space Signatures

Define four-character-sequences associated with color spaces.

```
enum {
    cmXYZData = 'XYZ ',
    cmLabData = 'Lab ',
    cmLuvData = 'Luv ',
    cmYCbCrData = 'YCbr',
    cmYxyData = 'Yxy ',
    cmRGBData = 'RGB ',
    cmSRGBData = 'sRGB',
    cmGrayData = 'GRAY',
    cmHSVData = 'HSV ',
    cmHLSData = 'HLS ',
    cmCMYKData = 'CMYK',
    cmCMYData = 'CMY ',
    cmMCH5Data = 'MCH5',
    cmMCH6Data = 'MCH6',
    cmMCH7Data = 'MCH7',
    cmMCH8Data = 'MCH8',
    cm3CLRData = '3CLR',
    cm4CLRData = '4CLR',
    cm5CLRData = '5CLR',
    cm6CLRData = '6CLR',
    cm7CLRData = '7CLR',
    cm8CLRData = '8CLR',
    cm9CLRData = '9CLR',
    cm10CLRData = 'ACLR',
    cm11CLRData = 'BCLR',
    cm12CLRData = 'CCLR',
    cm13CLRData = 'DCLR',
    cm14CLRData = 'ECLR',
    cm15CLRData = 'FCLR',
    cmNamedData = 'NAME'
};
```

Constants

`cmXYZData`

The XYZ data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmLabData`

The L*a*b* data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmLuvData`

The L*u*v* data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmYCbCrData`

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmYxyData`
The Yxy data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmRGBData`
The RGB data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSRGBData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGrayData`
The Gray data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmHSVData`
The HSV data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmHLSData`
The HLS data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCMYKData`
The CMYK data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCMYData`
The CMY data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH5Data`
The five-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH6Data`
The six-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH7Data`
The seven-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH8Data`
The eight-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm3CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm4CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm5CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm6CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm7CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm8CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm9CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm10CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm11CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm12CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm13CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm14CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm15CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

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

Declared in `CMICCProfile.h`.

Discussion
A ColorSync profile header contains a `dataColorSpace` field that carries the signature of the data color space in which the color values in an image using the profile are expressed. This enumeration defines the signatures for the color spaces supported by ColorSync for version 2.x profiles.

Color Space Masks

Specify masks used for color spaces.

```
enum {
    cmColorSpaceSpaceMask = 0x0000003F,
    cmColorSpacePremulAlphaMask = 0x00000040,
    cmColorSpaceAlphaMask = 0x00000080,
    cmColorSpaceSpaceAndAlphaMask = 0x000000FF,
    cmColorSpacePackingMask = 0x0000FF00,
    cmColorSpaceEncodingMask = 0x000F0000,
    cmColorSpaceReservedMask = 0xFFF00000
};
```

Constants
`cmColorSpaceSpaceMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

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

Declared in `CMApplication.h`.

ColorSync Scripting AppleEvent Errorsl

Define ColorSync AppleEvent scripting errors.

```
enum {
    cmspInvalidImageFile = -4220,
    cmspInvalidImageSpace = -4221,
    cmspInvalidProfileEmbed = -4222,
    cmspInvalidProfileSource = -4223,
    cmspInvalidProfileDest = -4224,
    cmspInvalidProfileProof = -4225,
    cmspInvalidProfileLink = -4226
};
```

Constants

`cmspInvalidImageFile`
Plugin cannot handle this image file type

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidImageSpace`
Plugin cannot create an image file of this colorspace

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidProfileEmbed`
Specific invalid profile errors

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

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

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

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

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

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

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

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

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

Current Device Versions

Specify the current versions of the data structure containing information on registered devices.

```
enum {
    cmDeviceInfoVersion1 = 0x00010000,
    cmDeviceProfileInfoVersion1 = 0x00010000,
    cmDeviceProfileInfoVersion2 = 0x00020000
};
```

Constants

`cmDeviceInfoVersion1`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceProfileInfoVersion1`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceProfileInfoVersion2`
Available in Mac OS X v10.1 and later.
Declared in `CMDeviceIntegration.h`.

Current Info Versions

Specify current device and profile versions.

```
enum {
    cmCurrentDeviceInfoVersion = cmDeviceInfoVersion1,
    cmCurrentProfileInfoVersion = cmDeviceProfileInfoVersion1
};
```

Constants

`cmCurrentDeviceInfoVersion`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmCurrentProfileInfoVersion`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

Current Major Version Mask

Specifies the current major version number.

```
enum {
    cmProfileMajorVersionMask = 0xFF000000,
    cmCurrentProfileMajorVersion = 0x02000000
};
```

Constants

`cmProfileMajorVersionMask`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCurrentProfileMajorVersion`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Data Transfer Commands

Specify commands for caller-supplied ColorSync data transfer functions.

```
enum {
    cmOpenReadSpool = 1,
    cmOpenWriteSpool = 2,
    cmReadSpool = 3,
    cmWriteSpool = 4,
    cmCloseSpool = 5
};
```

Constants

`cmOpenReadSpool`
Directs the function to begin the process of reading data.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmOpenWriteSpool`
Directs the function to begin the process of writing data.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmReadSpool`
Directs the function to read the number of bytes specified by the `CMFlattenProcPtr` function's `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmWriteSpool`
Directs the function to write the number of bytes specified by the `CMFlattenProcPtr` function's `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCloseSpool`
Directs the function to complete the data transfer.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion
When your application calls the function `CMFlattenProfile` (page 286), any of the functions in the group "Accessing Profile Elements" (page 17), or the PostScript-related functions of type "Working With PostScript" (page 20), the selected CMM—or, for the `CMUnflattenProfile` function, the ColorSync Manager—calls the flatten function you supply to transform profile data. The call passes one of the command constants defined by this enumeration.

Your application provides a pointer to your ColorSync data transfer function as a parameter to the functions. The ColorSync Manager or the CMM calls your data transfer function, passing the command in the `command` parameter. For more information on the flatten function, see `CMFlattenProfile` (page 286).

Data Type Element Values

Specify a data type.

```
enum {
    cmAsciiData = 0,
    cmBinaryData = 1
};
```

Constants
`cmAsciiData`
ASCII data.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBinaryData`
Binary data.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Default CMM Signature

Specifies a signature for the default color management module supplied by Color Sync.

```
enum {
    kDefaultCMMSignature = 'appl'
};
```

Constants
`kDefaultCMMSignature`
Signature for the default CMM supplied with the ColorSync Manager.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion
A color management module (CMM) uses profiles to convert and match a color in a given color space on a given device to or from another color space or device.

To specify the default CMM, set the `CMMType` field of the profile header to the default signature defined by the following enumeration. You use a structure of type `CM2Header` (page 116) for a ColorSync 2.x profile and a structure of type `CMHeader` (page 139) for a 1.0 profile header.

Default IDs

Specify default values for device and profile IDs.

```
enum {
    cmDefaultDeviceID = 0,
    cmDefaultProfileID = 0
};
```

Constants

`cmDefaultDeviceID`

Available in Mac OS X v10.0 and later.

Declared in `CMDeviceIntegration.h`.

`cmDefaultProfileID`

Available in Mac OS X v10.0 and later.

Declared in `CMDeviceIntegration.h`.

Discussion

Certain routines require a device ID or profile ID. In some cases, a "default ID" can be used.

Device Attribute Values for Version 2.x Profiles

Define masks your application can use to set or test bits in the `deviceAttributes` field of the `CM2Header` structure.

```
enum {
    cmReflectiveTransparentMask = 0x00000001,
    cmGlossyMatteMask = 0x00000002
};
```

Constants

`cmReflectiveTransparentMask`

Bit 0 of `deviceAttributes[1]` specifies whether the media is transparent or reflective. If it has the value 0, the media is reflective; if it has the value 1, the media is transparent. Use the `cmReflectiveTransparentMask` mask to set the transparent/reflective bit in `deviceAttributes[1]` or to clear all bits except the transparent/reflective bit.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmGlossyMatteMask`

Bit 1of `deviceAttributes[1]` specifies whether the media is glossy or matte. If it has the value 0, the media is glossy; if it has the value 1, the media is matte. Use the `cmGlossyMatteMask` mask to set the glossy/matte bit in `deviceAttributes[1]` or to clear all bits except the glossy/matte bit.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion

The ColorSync Manager defines the structure `CM2Header` (page 116) to represent the profile header for the version 2.x profile format defined by the ICC.The `deviceAttributes` field of the `CM2Header` structure is an array of two unsigned long values whose bits specify information about a profile. The ICC reserves the use of `deviceAttributes[1]` and has assigned values to bits 0 and 1. All the bits of `deviceAttributes[0]` are reserved for use by color management system (CMS) vendors.

Device Classes

Define constants to represent a variey of input and output devices.

```
enum {
    cmScannerDeviceClass = 'scnr',
    cmCameraDeviceClass = 'cmra',
    cmDisplayDeviceClass = 'mntr',
    cmPrinterDeviceClass = 'prtr',
    cmProofDeviceClass = 'pruf'
};
typedef OSType CMDeviceClass;
```

Constants

`cmScannerDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmCameraDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDisplayDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmPrinterDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmProofDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

Device and Media Attributes

Used to set or obtaind device or media attributes.

```
enum {
    cmReflective = 0,
    cmGlossy = 1
};
```

Constants

`cmReflective`
If the bit 0 of the associated mask is `0` then reflective media; if `1` then transparency media.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGlossy`
If the bit 1 of the associated mask is is `0` then glossy; if `1` then matte.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Device States

Specify device states.

```
enum {
    cmDeviceStateDefault = 0x00000000,
    cmDeviceStateOffline = 0x00000001,
    cmDeviceStateBusy = 0x00000002,
    cmDeviceStateForceNotify = 0x80000000,
    cmDeviceStateDeviceRsvdBits = 0x00FF0000,
    cmDeviceStateAppleRsvdBits = 0xFF00FFFF
};
```

Constants

`cmDeviceStateDefault`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateOffline`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateBusy`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateForceNotify`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateDeviceRsvdBits`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateAppleRsvdBits`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

Discussion

Specify possible values for device states accessible by the functions `CMGetDeviceState` and `CMSetDeviceState`.

Device Types

Specify a device type.

```
enum {
    cmMonitorDevice = 'mntr',
    cmScannerDevice = 'scnr',
    cmPrinterDevice = 'prtr'
};
```

Constants

`cmMonitorDevice`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmScannerDevice`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmPrinterDevice`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

Element Tags and Signatures for Version 1.0 Profiles

Define tags and signatures used for version 1.0 profiles.

```
enum {
    cmCS1ChromTag = 'chrm',
    cmCS1TRCTag = 'trc ',
    cmCS1NameTag = 'name',
    cmCS1CustTag = 'cust'
};
```

Constants

`cmCS1ChromTag`

The tag signature for the profile chromaticities tag whose element data specifies the XYZ chromaticities for the six primary and secondary colors (red, green, blue, cyan, magenta, and yellow).

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1TRCTag`

The tag signature for profile tonal response curve data for the associated device.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1NameTag`

The tag signature for the profile name string. This is an international string consisting of a Macintosh script code followed by a 63-byte text string identifying the profile.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1CustTag`
Private data for a custom CMM.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion
The ICC version 2.x profile format differs from the version 1.0 profile format, and ColorSync Manager routines for updating a profile and searching for profiles do not work with version 1.0 profiles. However, your application can use version 1.0 profiles with all other ColorSync routines. For example, you can open a version 1.0 profile using the function `CMOpenProfile` (page 63), obtain the version 1.0 profile header using the function `CMGetProfileHeader` (page 47), and access version 1.0 profile elements using the function `CMGetProfileElement` (page 46).

To make this possible, the ColorSync Manager includes support for the version 1.0 profile header structure and synthesizes tags to allow you to access four 1.0 elements outside the version 1.0 profile header. This enumeration defines these tags.

Embedded Profile Flags

Specify copyright-protection flag options,

```
enum {
    cmEmbeddedProfile = 0,
    cmEmbeddedUse = 1
};
```

Constants
`cmEmbeddedProfile`
0 is not embedded profile, 1 is embedded profile

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmEmbeddedUse`
0 is to use anywhere, 1 is to use as embedded profile only

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Embedded Profile Identifiers

Specify constants used when embedding picture comments.

```
enum {
    cmEmbedWholeProfile = 0x00000000,
    cmEmbedProfileIdentifier = 0x00000001
};
```

Constants

`cmEmbedWholeProfile`

When the `flags` parameter has the value `cmEmbedWholeProfile`, the `NCMUseProfileComment` function embeds the entire specified profile.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEmbedProfileIdentifier`

When the `flags` parameter has the value `cmEmbedProfileIdentifier`, the `NCMUseProfileComment` function embeds a profile identifier for the specified profile.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Discussion

The ColorSync Manager provides these constant declarations to use with the function `NCMUseProfileComment` (page 272) for embedding picture comments. You use these constants to set the `flags` parameter to indicate whether to embed an entire profile or just a profile identifier.

Flag Mask Definitions for Version 2.x Profiles

Define masks your application can use to set or test various bits in the `flags` field of the `CM2Header` structure.

```
enum {
    cmICCReservedFlagsMask = 0x0000FFFF,
    cmEmbeddedMask = 0x00000001,
    cmEmbeddedUseMask = 0x00000002,
    cmCMSReservedFlagsMask = 0xFFFF0000,
    cmQualityMask = 0x00030000,
    cmInterpolationMask = 0x00040000,
    cmGamutCheckingMask = 0x00080000
};
```

Constants

`cmICCReservedFlagsMask`

This mask provides access to bits 0 through 15 of the `flags` field, which are defined and reserved by the ICC. For more information, see the International Color Consortium Profile Format Specification, and the next two mask definitions.

To obtain a copy of the ICC specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmEmbeddedMask`

This mask provides access to bit 0 of the `flags` field, which specifies whether the profile is embedded. It has the value 1 if the profile is embedded, 0 if it is not.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmEmbeddedUseMask`

This mask provides access to bit 1 of the `flags` field, which specifies whether the profile can be used independently or can only be used as an embedded profile. It has the value 0 if the profile can be used anywhere, 1 if it must be embedded.

You should interpret the setting of this bit as an indication of copyright protection. If the profile developer set this bit to 1, you should use this profile as an embedded profile only and not copy the profile for your own purposes. The profile developer also specifies explicit copyright intention using the `cmCopyrightTag` profile tag (defined in the `CMICCProfile.h` header file).

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCMSReservedFlagsMask`

This mask provides access to bits 16 through 31 of the `flags` field, which are available for a color management system (CMS) vendor, such as ColorSync. ColorSync's default CMM uses bits 16 through 19 to provide hints for color matching, as described in the following three mask definitions. Other CMM vendors should follow the same conventions.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmQualityMask`

This mask provides access to bits 16 and 17 of the `flags` field, which specify the preferred quality and speed preferences for color matching. In general, the higher the quality the slower the speed. For example, best quality is slowest, but produces the highest quality result.

Bits 16 and 17 have the value 0 for normal quality, 1 for draft quality, and 2 for best quality. "Quality Flag Values for Version 2.x Profiles" (page 252) describes the constants ColorSync defines to test or set these bits.

This feature is provided by the ColorSync Manager; it is not defined by the ICC profile specification.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmInterpolationMask`

This mask provides access to bit 18 of the `flags` field, which specifies whether to use interpolation in color matching. The value 0 specifies interpolation. The value 1 specifies table lookup without interpolation. Specifying lookup only improves speed but can reduce accuracy. You might use lookup only for a monitor profile, for example, when high resolution is not crucial.

This feature is provided by the ColorSync Manager; it is not defined by the ICC profile specification.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmGamutCheckingMask`

This mask provides access to bit 19 of the `flags` field. When you use a profile to create a color world, bit 19 specifies whether the color world should include information for gamut checking. It has the value 0 if the color world should include a gamut-checking table, 1 if gamut-checking information is not required. ColorSync can create a color world without a gamut table more quickly and in less space.

Many applications do not perform gamut checking, so they should set this bit to 1. However, if you call a color checking function such as `CWCheckColors` (page 81), or `CWMatchColors` (page 87), after setting a profile's gamut-checking bit so that the color world does not contain gamut information, these routines return the `cmCantGamutCheckError` error.

This feature is provided by the ColorSync Manager; it is not defined by the ICC profile specification.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion

The `flags` field of the structure `CM2Header` (page 116) is an unsigned long value whose bits specify information about a profile. The ICC reserves the use of bits 0 to 15 and has assigned values to bits 0 and 1. Bits 16 to 31 are reserved for use by color management system (CMS) vendors. ColorSync has assigned values to bits 16 through 19.

ICC Profile Versions

Specify IDD profile version numbers.

```
enum {
    cmICCProfileVersion4 = 0x04000000,
    cmICCProfileVersion2 = 0x02000000,
    cmICCProfileVersion21 = 0x02100000,
    cmCS2ProfileVersion = cmICCProfileVersion2,
    cmCS1ProfileVersion = 0x00000100
};
```

Constants

`cmICCProfileVersion4`

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmICCProfileVersion2`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmICCProfileVersion21`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS2ProfileVersion`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1ProfileVersion`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Illuminant Measurement Endocings

Specify standard illuminate measurement encodings.

```
enum {
    cmIlluminantUnknown = 0x00000000,
    cmIlluminantD50 = 0x00000001,
    cmIlluminantD65 = 0x00000002,
    cmIlluminantD93 = 0x00000003,
    cmIlluminantF2 = 0x00000004,
    cmIlluminantD55 = 0x00000005,
    cmIlluminantA = 0x00000006,
    cmIlluminantEquiPower = 0x00000007,
    cmIlluminantF8 = 0x00000008
};
```

Constants

`cmIlluminantUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD50`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD65`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD93`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantF2`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD55`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantA`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantEquiPower`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantF8`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Macintosh 68K Trap Word

Specifies a 68K trap word for the Macintosh.

```
enum {
    cmTrap = 0xABEE
};
```

Constants

`cmTrap`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMApplication.h`.

Magic Cookie Number

Specifies a magic cookie number for anonymous file ID.

```
enum {
    cmMagicNumber = 'acsp'
};
```

Constants

`cmMagicNumber`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Match Flags Field

Specifies a profile to match.

```
enum {
    cmspFavorEmbeddedMask = 0x00000001
};
```

Constants

`cmspFavorEmbeddedMask`

If bit 0 is `0` then use `srcProf` profile; if `1` then use profile embedded in image if present.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

Match Profiles 2.0

Defines matching flags for version 2.0 of the `CMSearchRecord.searchMask`.

```
enum {
    cmMatchAnyProfile = 0x00000000,
    cmMatchProfileCMMType = 0x00000001,
    cmMatchProfileClass = 0x00000002,
    cmMatchDataColorSpace = 0x00000004,
    cmMatchProfileConnectionSpace = 0x00000008,
    cmMatchManufacturer = 0x00000010,
    cmMatchModel = 0x00000020,
    cmMatchAttributes = 0x00000040,
    cmMatchProfileFlags = 0x00000080
};
```

Constants

`cmMatchAnyProfile`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileCMMType`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileClass`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchDataColorSpace`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileConnectionSpace`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchManufacturer`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchModel`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchAttributes`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileFlags`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Match Profiles 1.0

Defines matching flags for version 1.0 of the `CMSearchRecord.searchMask`.

```
enum {
    cmMatchCMMType = 0x00000001,
    cmMatchApplProfileVersion = 0x00000002,
    cmMatchDataType = 0x00000004,
    cmMatchDeviceType = 0x00000008,
    cmMatchDeviceManufacturer = 0x00000010,
    cmMatchDeviceModel = 0x00000020,
    cmMatchDeviceAttributes = 0x00000040,
    cmMatchFlags = 0x00000080,
    cmMatchOptions = 0x00000100,
    cmMatchWhite = 0x00000200,
    cmMatchBlack = 0x00000400
};
```

Constants

`cmMatchCMMType`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchApplProfileVersion`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDataType`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDeviceType`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDeviceManufacturer`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDeviceModel`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchDeviceAttributes`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchFlags`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchOptions`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchWhite`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchBlack`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

Maximum Path Size

Specifies the maximum length for a path name.

```
enum {
    CS_MAX_PATH = 256
};
```

Constants

`CS_MAX_PATH`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Measurement Flares

Specify measurement flare encodings.

```
enum {
    cmFlare0 = 0x00000000,
    cmFlare100 = 0x00000001
};
```

Constants

`cmFlare0`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmFlare100`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Measurment Geometries

Specify measurement geometry encodings.

```
enum {
    cmGeometryUnknown = 0x00000000,
    cmGeometry045or450 = 0x00000001,
    cmGeometry0dord0 = 0x00000002
};
```

Constants

`cmGeometryUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGeometry045or450`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGeometry0dord0`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Obsolete Color Response Values

Redefines obsolete color response values.

```
enum {
    grayResponse = cmGrayResponse,
    redResponse = cmRedResponse,
    greenResponse = cmGreenResponse,
    blueResponse = cmBlueResponse,
    cyanResponse = cmCyanResponse,
    magentaResponse = cmMagentaResponse,
    yellowResponse = cmYellowResponse,
    ucrResponse = cmUcrResponse,
    bgResponse = cmBgResponse,
    onePlusLastResponse = cmOnePlusLastResponse
};
```

Obsolete Color Space Signatures

Redefines obsolete color space signatures.

```
enum {
    rgbData = cmRGBData,
    cmykData = cmCMYKData,
    grayData = cmGrayData,
    xyzData = cmXYZData
};
```

Obsolete Device Type Names

Redefines obsolete device type names.

```
enum {
    monitorDevice = cmMonitorDevice,
    scannerDevice = cmScannerDevice,
    printerDevice = cmPrinterDevice
};
```

Parametric Types

Specify a parametric curve type enumeration,

```
enum {
    cmParametricType0 = 0,
    cmParametricType1 = 1,
    cmParametricType2 = 2,
    cmParametricType3 = 3,
    cmParametricType4 = 4
};
```

Constants

`cmParametricType0`

Y = X^gamma

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType1`
Y = (aX+b)^gamma [X>=-b/a], Y = 0 [X<-b/a]

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType2`
Y = (aX+b)^gamma + c [X>=-b/a], Y = c [X<-b/a]

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType3`
Y = (aX+b)^gamma [X>=d], Y = cX [X<d]

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType4`
Y = (aX+b)^gamma + e [X>=d], Y = cX+f [X<d]

Available in Mac OS X v10.2 and later.

Declared in `CMICCProfile.h`.

Platform Enumeration Values

Specify computer platforms.

```
enum {
    cmMacintosh = 'APPL',
    cmMicrosoft = 'MSFT',
    cmSolaris = 'SUNW',
    cmSiliconGraphics = 'SGI ',
    cmTaligent = 'TGNT'
};
```

Profile Iteration Values

Specify profiles to iterate.

```
enum {
    cmIterateFactoryDeviceProfiles = 0x00000001,
    cmIterateCustomDeviceProfiles = 0x00000002,
    cmIterateCurrentDeviceProfiles = 0x00000003,
    cmIterateAllDeviceProfiles = 0x00000004,
    cmIterateDeviceProfilesMask = 0x0000000F
};
```

Constants

`cmIterateFactoryDeviceProfiles`
Iterate profiles registered through the routine `CMSetDeviceFactoryProfiles`. To retrieve all factory profiles for all devices, use `cmIterateFactoryDeviceProfiles` as the flags value when calling `CMIterateDeviceProfiles`. I

Available in Mac OS X v10.0 and later.

Declared in `CMDeviceIntegration.h`.

`cmIterateCustomDeviceProfiles`
: Iterate profiles that are meant to take the place of the factory profiles, as a result of customization or calibration. To retrieve only custom profiles for all devices, use the `cmIterateCustomDeviceProfiles`, as the flags value when calling `CMIterateDeviceProfiles`.

  Available in Mac OS X v10.0 and later.

  Declared in `CMDeviceIntegration.h`.

`cmIterateCurrentDeviceProfiles`
: Iterate profiles registered through the routing `CMSetDeviceProfiles`. To get the profiles in use for all devices, use `cmIterateCurrentDeviceProfiles` as the flags value. This will replace the factory profiles with any overrides, yielding the currently used set.l

  Available in Mac OS X v10.0 and later.

  Declared in `CMDeviceIntegration.h`.

`cmIterateAllDeviceProfiles`
: Iterate all profiles, without replacement.

  Available in Mac OS X v10.1 and later.

  Declared in `CMDeviceIntegration.h`.

`cmIterateDeviceProfilesMask`
: Available in Mac OS X v10.0 and later.

  Declared in `CMDeviceIntegration.h`.

Discussion
These are possible values for flags passed to the function `CMIterateDeviceProfiles`.

Profile Location Sizes

Specify a location size.

```
enum {
    cmOriginalProfileLocationSize = 72,
    cmCurrentProfileLocationSize = 2 + CS_MAX_PATH
};
```

Constants
`cmOriginalProfileLocationSize`
: Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmCurrentProfileLocationSize`
: Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

Profile Options

Specify a rendering intent.

```
enum {
    cmPerceptualMatch = 0x0000,
    cmColorimetricMatch = 0x0001,
    cmSaturationMatch = 0x0002
};
```

Constants

`cmPerceptualMatch`
Default. For photographic images
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmColorimetricMatch`
Exact matching when possible
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmSaturationMatch`
For solid colors
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

PostScript Data Formats

Specify constants that indicate the format of PostScript data.

```
enum {
    cmPS7bit = 1,
    cmPS8bit = 2
};
```

Constants

`cmPS7bit`
The data is 7-bit safe—therefore the data could be in 7-bit ASCII encoding or in ASCII base-85 encoding.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmPS8bit`
The data is 8-bit safe—therefore the data could be in 7-bit or 8-bit ASCII encoding.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion

The ColorSync Manager provides these constant declarations to specify the format of PostScript data.

Picture Comment Kinds

Specify picture comment kinds for profiles and color matching.

```
enum {
    cmBeginProfile = 220,
    cmEndProfile = 221,
    cmEnableMatching = 222,
    cmDisableMatching = 223,
    cmComment = 224
};
```

Constants

`cmBeginProfile`

Indicates the beginning of a version 1.0 profile to embed. (To start a 2.x profile, you use `cmComment`.)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEndProfile`

Signals end of the use of an embedded version 2.x or 1.0 profile.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEnableMatching`

Turns on color matching for the ColorSync Manager. Do not nest `cmEnableMatching` and `cmDisableMatching` pairs.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmDisableMatching`

Turns off color matching for the ColorSync Manager. Do not nest `cmEnableMatching` and `cmDisableMatching` pairs. After the ColorSync Manager encounters this comment, it ignores all ColorSync-related picture comments until it encounters the next `cmEnableMatching` picture comment. At that point, the most recently used profile is reinstated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmComment`

Provides information about a 2.x embedded profile or embedded profile identifier reference. This picture comment is followed by a 4-byte selector identifying what follows. "Picture Comment Selectors" (page 238) describes the possible selectors.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Discussion

The ColorSync Manager defines five picture comment kinds. You use these comments to embed a profile identifier, begin or end use of an embedded profile, and enable or disable color matching within drawing code sent to an output device. The `PicComment` function's `kind` parameter specifies the kind of picture comment.

Use a picture comment of kind `cmEndProfile` to explicitly terminate use of the currently effective embedded profile and begin use of the system profile. Otherwise, the currently effective profile remains in effect, leading to unexpected results if another picture follows that is meant to use the system profile and so is not preceded by a profile.

Picture Comment Selectors

Specify selectors to use in piture comments.

```
enum {
    cmBeginProfileSel = 0,
    cmContinueProfileSel = 1,
    cmEndProfileSel = 2,
    cmProfileIdentifierSel = 3
};
```

Constants

`cmBeginProfileSel`

Identifies the beginning of version 2.x profile data. The amount of profile data you can specify is limited to 32K minus 4 bytes for the selector.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmContinueProfileSel`

Identifies the continuation of version 2.x profile data. The amount of profile data you can specify is limited to 32K minus 4 bytes for the selector. You can use this selector repeatedly until all the profile data is embedded.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEndProfileSel`

Signals the end of version 2.x profile data—no more data follows. Even if the amount of profile data embedded does not exceed 32K minus 4 bytes for the selector and your application did not use `cmContinueProfileSel`, you must terminate the process with `cmEndProfileSel`. Note that this selector has a behavior that is different from the `cmEndProfile` picture comment described in "Picture Comment Kinds" (page 236).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmProfileIdentifierSel`

Identifies the inclusion of profile identifier data. For information on embedding a profile identifier, see the function `NCMUseProfileComment` (page 272). For information on the format of profile identifier data, see `CMProfileIdentifier` (page 162).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Discussion
To embed a version 2.x profile or profile identifier reference in a picture destined for display on another system or on a device such as a printer, your application uses the QuickDraw `PicComment` function. The ColorSync Manager provides the function `NCMUseProfileComment` (page 272) to embed picture comments. You specify a picture comment `kind` value of `cmComment` and a 4-byte selector describing the data in the picture comment.

Because a profile may exceed QuickDraw's 32 KB size limit for a picture comment, your application can use an ordered series of picture comments to embed a large profile.

You can also embed a profile identifier reference in a picture. The profile identifier may refer to a previously embedded profile, so that you do not have to embed the entire profile again, or it may refer to a profile stored on disk. When you embed a profile identifier, you can change certain values for the referred-to profile, including the quality flags and rendering intent. For more information on profile identifiers, see `CMProfileIdentifier` (page 162).

This enumeration defines the 4-byte selector values your application uses to identify the beginning and continuation of profile data and to signal the end of it.

Profile Access Procedures

Specify operations used to access profiles.

```
enum {
    cmOpenReadAccess = 1,
    cmOpenWriteAccess = 2,
    cmReadAccess = 3,
    cmWriteAccess = 4,
    cmCloseAccess = 5,
    cmCreateNewAccess = 6,
    cmAbortWriteAccess = 7,
    cmBeginAccess = 8,
    cmEndAccess = 9
};
```

Constants

`cmOpenReadAccess`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmOpenWriteAccess`
Open the profile for writing. The total size of the profile is specified in the `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmReadAccess`
Read the number of bytes specified by the `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmWriteAccess`
Write the number of bytes specified by the `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCloseAccess`
: Close the profile for reading or writing.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmCreateNewAccess`
: Create a new data stream for the profile.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmAbortWriteAccess`
: Cancel the current write attempt.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmBeginAccess`
: Begin the process of procedural access. This is always the first operation constant passed to the access procedure. If the call is successful, the `cmEndAccess` operation is guaranteed to be the last call to the procedure.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmEndAccess`
: End the process of procedural access. This is always the last operation constant passed to the access procedure (unless the `cmBeginAccess` call failed).

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

Discussion
When your application calls the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, or `CMNewLinkProfile` functions, it can supply the ColorSync Manager with a profile location structure of type `CMProcedureLocation` (page 160) to specify a procedure that provides access to a profile. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed. The profile access procedure declaration is described in `CMProfileAccessProcPtr` (page 103).

When the ColorSync Manager calls your profile access procedure, it passes one of these constants in the `command` parameter to specify an operation. Your procedure must be able to respond to each of these constants.

Profile Classes

Specify profile class enumerations.

```
enum {
    cmInputClass = 'scnr',
    cmDisplayClass = 'mntr',
    cmOutputClass = 'prtr',
    cmLinkClass = 'link',
    cmAbstractClass = 'abst',
    cmColorSpaceClass = 'spac',
    cmNamedColorClass = 'nmcl'
};
```

Constants

`cmInputClass`
An input device profile defined for a scanner.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDisplayClass`
A display device profile defined for a monitor.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmOutputClass`
An output device profile defined for a printer.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmLinkClass`
A device link profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmAbstractClass`
An abstract profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmColorSpaceClass`
A color space profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNamedColorClass`
A named color space profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Discussion

The ColorSync Manager supports seven classes, or types, of profiles.

A profile creator specifies the profile class in the profile header's `profileClass` field. For a description of the profile header, see `CM2Header` (page 116). This enumeration defines the profile class signatures.

Profile Concatenation Values

Specify values to use when concatnating profiles.

```
enum {
    kNoTransform = 0,
    kUseAtoB = 1,
    kUseBtoA = 2,
    kUseBtoB = 3,
    kDeviceToPCS = kUseAtoB,
    kPCSToDevice = kUseBtoA,
    kPCSToPCS = kUseBtoB,
    kUseProfileIntent = 0xFFFFFFFF
};
```

Constants

`kNoTransform`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseAtoB`
Use 'A2B*' tag from this profile or equivalent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseBtoA`
Use 'B2A*' tag from this profile or equivalent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseBtoB`
Use 'pre*' tag from this profile or equivalent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kDeviceToPCS`
Device Dependent to Device Independent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kPCSToDevice`
Device Independent to Device Dependent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kPCSToPCS`
Independent, through device's gamut
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseProfileIntent`
For renderingIntent in NCMConcatProfileSpec

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Profile Flags

Define flags that control native matchign and caching.

```
enum {
    cmNativeMatchingPreferred = 0x00000001,
    cmTurnOffCache = 0x00000002
};
```

Constants

`cmNativeMatchingPreferred`
Default to native not preferred

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmTurnOffCache`
Default to turn on CMM cache

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

Profile Iteration Constants

Define an interation version.

```
enum {
    cmProfileIterateDataVersion1 = 0x00010000,
    cmProfileIterateDataVersion2 = 0x00020000,
    cmProfileIterateDataVersion3 = 0x00030000
};
```

Constants

`cmProfileIterateDataVersion1`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmProfileIterateDataVersion2`
Added makeAndModel

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmProfileIterateDataVersion3`
Added MD5 digest

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Profile Location Type

Defines profile location kinds.

```
enum {
    cmNoProfileBase = 0,
    cmFileBasedProfile = 1,
    cmHandleBasedProfile = 2,
    cmPtrBasedProfile = 3,
    cmProcedureBasedProfile = 4,
    cmPathBasedProfile = 5,
    cmBufferBasedProfile = 6
};
```

Constants

`cmNoProfileBase`
The profile is temporary. It will not persist in memory after its use for a color session. You can specify this type of profile location with the `CMNewProfile` and the `CMCopyProfile` functions.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmFileBasedProfile`
The profile is stored in a disk-file and the `CMProfLoc` union of type `CMProfLoc` (page 169) holds a structure of type `CMFileLocation` (page 137) identifying the profile file. You can specify this type of profile location with the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, and `CMNewLinkProfile` functions.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmHandleBasedProfile`
The profile is stored in relocatable memory and the `CMProfLoc` union of type `CMProfLoc` (page 169) holds a handle to the profile in a structure of type `CMHandleLocation` (page 139). You can specify this type of profile location with the `CMOpenProfile`, `CMNewProfile`, and `CMCopyProfile` functions.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmPtrBasedProfile`
The profile is stored in nonrelocatable memory and the `CMProfLoc` union of type `CMProfLoc` (page 169) holds a pointer to the profile in a structure of type `CMPtrLocation` (page 170). You can specify this type of profile location with the `CMOpenProfile` function only.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmProcedureBasedProfile`
: The profile is in an arbitrary location, accessed through a procedure supplied by you. The `CMProfLoc` union of type `CMProfLoc` (page 169) holds a universal procedure pointer to your profile access procedure in a structure of type `CMProcedureLocation` (page 160). You can specify this type of profile location with the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, and `CMNewLinkProfile` functions. For a description of an application-supplied profile access procedure, see `CMProfileAccessProcPtr` (page 103).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmPathBasedProfile`
: Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmBufferBasedProfile`
: Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion
Your application specifies the location for a profile using a profile location structure of type `CMProfileLocation` (page 165). A ColorSync profile that you open or create is typically stored in one of the following locations:

- In a disk file. The `u` field (a union) of the profile location data structure contains a file specification for a profile that is disk-file based. This is the most common way to store a ColorSync profile.
- In relocatable memory. The `u` field of the profile location data structure contains a handle specification for a profile that is stored in a handle.
- In nonrelocatable memory. The `u` field of the profile location data structure contains a pointer specification for a profile that is pointer based.
- In an arbitrary location, accessed by a procedure you provide. The `u` field of the profile location data structure contains a universal procedure pointer to your access procedure, as well as a pointer that may point to data associated with your procedure.

Additionally, your application can create a new or duplicate temporary profile. For example, you can use a temporary profile for a color-matching session and the profile is not saved after the session. For this case, the ColorSync Manager allows you to specify the profile location as having no specific location.

You use a pointer to a data structure of type `CMProfileLocation` to identify a profile's location when your application calls

- the `CMOpenProfile` function to obtain a reference to a profile
- the `CMNewProfile`, `CWNewLinkProfile`, or `CMCopyProfile` functions to create a new profile
- the `CMGetProfileLocation` function to get the location of an existing profile

Your application identifies the type of data the `CMProfileLocation u` field holds—a file specification, a handle, and so on—in the `CMProfileLocation` structure's `locType` field. You use the constants defined by this enumeration to identify the location type.

Public Tags

Specify tag values available for public use.

```
enum {
    cmAToB0Tag = 'A2B0',
    cmAToB1Tag = 'A2B1',
    cmAToB2Tag = 'A2B2',
    cmBlueColorantTag = 'bXYZ',
    cmBlueTRCTag = 'bTRC',
    cmBToA0Tag = 'B2A0',
    cmBToA1Tag = 'B2A1',
    cmBToA2Tag = 'B2A2',
    cmCalibrationDateTimeTag = 'calt',
    cmChromaticAdaptationTag = 'chad',
    cmCharTargetTag = 'targ',
    cmCopyrightTag = 'cprt',
    cmDeviceMfgDescTag = 'dmnd',
    cmDeviceModelDescTag = 'dmdd',
    cmGamutTag = 'gamt',
    cmGrayTRCTag = 'kTRC',
    cmGreenColorantTag = 'gXYZ',
    cmGreenTRCTag = 'gTRC',
    cmLuminanceTag = 'lumi',
    cmMeasurementTag = 'meas',
    cmMediaBlackPointTag = 'bkpt',
    cmMediaWhitePointTag = 'wtpt',
    cmNamedColorTag = 'ncol',
    cmNamedColor2Tag = 'ncl2',
    cmPreview0Tag = 'pre0',
    cmPreview1Tag = 'pre1',
    cmPreview2Tag = 'pre2',
    cmProfileDescriptionTag = 'desc',
    cmProfileSequenceDescTag = 'pseq',
    cmPS2CRD0Tag = 'psd0',
    cmPS2CRD1Tag = 'psd1',
    cmPS2CRD2Tag = 'psd2',
    cmPS2CRD3Tag = 'psd3',
    cmPS2CSATag = 'ps2s',
    cmPS2RenderingIntentTag = 'ps2i',
    cmRedColorantTag = 'rXYZ',
    cmRedTRCTag = 'rTRC',
    cmScreeningDescTag = 'scrd',
    cmScreeningTag = 'scrn',
    cmTechnologyTag = 'tech',
    cmUcrBgTag = 'bfd ',
    cmViewingConditionsDescTag = 'vued',
    cmViewingConditionsTag = 'view'
};
```

Constants

`cmAToB0Tag`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmAToB1Tag`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmAToB2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBlueColorantTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBlueTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBToA0Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBToA1Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBToA2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCalibrationDateTimeTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmChromaticAdaptationTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCharTargetTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCopyrightTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDeviceMfgDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDeviceModelDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGamutTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGrayTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGreenColorantTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGreenTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmLuminanceTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMeasurementTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMediaBlackPointTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMediaWhitePointTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNamedColorTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNamedColor2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPreview0Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPreview1Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPreview2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmProfileDescriptionTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmProfileSequenceDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD0Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD1Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD3Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CSATag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2RenderingIntentTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmRedColorantTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmRedTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmScreeningDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmScreeningTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmUcrBgTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmViewingConditionsDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmViewingConditionsTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Public Type Signatures

Specify signatures for public types.

```
enum {
    cmSigCrdInfoType = 'crdi',
    cmSigCurveType = 'curv',
    cmSigDataType = 'data',
    cmSigDateTimeType = 'dtim',
    cmSigLut16Type = 'mft2',
    cmSigLut8Type = 'mft1',
    cmSigMeasurementType = 'meas',
    cmSigMultiFunctA2BType = 'mAB ',
    cmSigMultiFunctB2AType = 'mBA ',
    cmSigNamedColorType = 'ncol',
    cmSigNamedColor2Type = 'ncl2',
    cmSigParametricCurveType = 'para',
    cmSigProfileDescriptionType = 'desc',
    cmSigProfileSequenceDescType = 'pseq',
    cmSigScreeningType = 'scrn',
    cmSigS15Fixed16Type = 'sf32',
    cmSigSignatureType = 'sig ',
    cmSigTextType = 'text',
    cmSigU16Fixed16Type = 'uf32',
    cmSigU1Fixed15Type = 'uf16',
    cmSigUInt8Type = 'ui08',
    cmSigUInt16Type = 'ui16',
    cmSigUInt32Type = 'ui32',
    cmSigUInt64Type = 'ui64',
    cmSigUcrBgType = 'bfd ',
    cmSigUnicodeTextType = 'utxt',
    cmSigViewingConditionsType = 'view',
    cmSigXYZType = 'XYZ '
};
```

Constants

`cmSigCrdInfoType`

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmSigCurveType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigDataType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigDateTimeType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigLut16Type`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigLut8Type`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigMeasurementType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigMultiFunctA2BType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigMultiFunctB2AType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigNamedColorType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigNamedColor2Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigParametricCurveType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigProfileDescriptionType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigProfileSequenceDescType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigScreeningType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigS15Fixed16Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigSignatureType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigTextType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigU16Fixed16Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigU1Fixed15Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUInt8Type`
Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigUInt16Type`
Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigUInt32Type`
Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigUInt64Type`
Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

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

Declared in `CMICCProfile.h`.

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

Declared in `CMICCProfile.h`.

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

Declared in `CMICCProfile.h`.

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

Declared in `CMICCProfile.h`.

Quality Flag Values for Version 2.x Profiles

Define the possible values for the quality bits in the `flags` field of the `CM2Header` structure.

```
enum {
    cmNormalMode = 0,
    cmDraftMode = 1,
    cmBestMode = 2
};
```

Constants

`cmNormalMode`
This is the default setting. Normal mode indicates that the CMM should use its default method to compromise between performance and resource requirements.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmDraftMode`

Draft mode indicates that the CMM should sacrifice quality, if necessary, to minimize resource requirements. Note that the default CMM currently produces the same results for both normal and draft mode.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmBestMode`

Best mode indicates that the CMM should maximize resource usage to ensure the highest possible quality.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion

To determine the value of the quality flag, you mask the `flags` field of the profile header with the `cmQualityMask` mask, right shift 16 bits, then compare the result to the enumerated constants shown here. For more information on the quality flag, see "Flag Mask Definitions for Version 2.x Profiles" (page 224).

When you start a color-matching session, ColorSync sends all involved profiles to the color management module (CMM). The CMM extracts the information it needs from the profiles and stores an internal representation in private memory. ColorSync's default CMM samples the input space and stores the results in a lookup table, a common technique that speeds up conversion for runtime applications. The size of the table is based on the quality flag setting in the source profile header. The setting of the quality flag can affect the memory requirements, accuracy, and speed of the color-matching session. In general, the higher the quality setting, the larger the lookup table, the more accurate the matching, and the slower the matching process. Note however, that the default CMM currently produces the same results for both normal and draft mode.

Rendering Intent Values for Version 2.x Profiles

Define the four possible values for the rendering intent bits of the `renderingIntent` field of the `CM2Header` structure.

```
enum {
    cmPerceptual = 0,
    cmRelativeColorimetric = 1,
    cmSaturation = 2,
    cmAbsoluteColorimetric = 3
};
```

Constants

`cmPerceptual`

All the colors of a given gamut can be scaled to fit within another gamut. This intent is best suited to realistic images, such as photographic images.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmRelativeColorimetric`

The colors that fall within the gamuts of both devices are left unchanged. This intent is best suited to logo images.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSaturation`
: The relative saturation of colors is maintained from gamut to gamut. This intent is best suited to bar graphs and pie charts in which the actual color displayed is less important than its vividness.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

`cmAbsoluteColorimetric`
: This approach is based on a device-independent color space in which the result is an idealized print viewed on a ideal type of paper having a large dynamic range and color gamut.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

Discussion
The ColorSync Manager defines the structure `CM2Header` (page 116) to represent the profile header for the version 2.x profile format defined by the ICC. The `renderingIntent` field of the `CM2Header` structure is an unsigned long value whose bits specify information about a profile. The ICC reserves the use of bits 0 to 15 and has assigned values to bits 0 and 1. Bits 16 to 31 are reserved for use by color management system (CMS) vendors.

Rendering intent controls the approach a CMM uses to translate the colors of an image to the color gamut of a destination device. Your application can set a profile's rendering intent, for example, based on a user's choice of the preferred approach for rendering an image.

Because rendering intent is specified by the low two bits, and because no other bits are currently defined for this field, you can use the constants defined here to test or set the value of the entire field, without concern for possible information stored in other bits.

Screen Encoding Tags

Specify tags to use for screen encodings.

```
enum {
    cmPrtrDefaultScreens = 0,
    cmLinesPer = 1
};
```

Constants
`cmPrtrDefaultScreens`
: Use printer default screens; can have an associated value of `0` for `false` or `1` for `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

`cmLinesPer`
: Lines per unit; can have an associated value of `0` for lines per centimeter or `1` for lines per inch.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

Spot Function Values

Speicfy values for spot functions.

```
enum {
    cmSpotFunctionUnknown = 0,
    cmSpotFunctionDefault = 1,
    cmSpotFunctionRound = 2,
    cmSpotFunctionDiamond = 3,
    cmSpotFunctionEllipse = 4,
    cmSpotFunctionLine = 5,
    cmSpotFunctionSquare = 6,
    cmSpotFunctionCross = 7
};
```

Constants

`cmSpotFunctionUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionDefault`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionRound`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionDiamond`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionEllipse`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionLine`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionSquare`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionCross`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Standard Oberver

Standard observer measurement type encodings.

```
enum {
    cmStdobsUnknown = 0x00000000,
    cmStdobs1931TwoDegrees = 0x00000001,
    cmStdobs1964TenDegrees = 0x00000002
};
```

Constants

`cmStdobsUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmStdobs1931TwoDegrees`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmStdobs1964TenDegrees`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Tag Type Information

Defines a constant for 2.0 tag type information.

```
enum {
    cmNumHeaderElements = 10
};
```

Constants

`cmNumHeaderElements`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Technology Tag Descriptions

Define descriptor tags for technologies.

```
enum {
    cmTechnologyDigitalCamera = 'dcam',
    cmTechnologyFilmScanner = 'fscn',
    cmTechnologyReflectiveScanner = 'rscn',
    cmTechnologyInkJetPrinter = 'ijet',
    cmTechnologyThermalWaxPrinter = 'twax',
    cmTechnologyElectrophotographicPrinter = 'epho',
    cmTechnologyElectrostaticPrinter = 'esta',
    cmTechnologyDyeSublimationPrinter = 'dsub',
    cmTechnologyPhotographicPaperPrinter = 'rpho',
    cmTechnologyFilmWriter = 'fprn',
    cmTechnologyVideoMonitor = 'vidm',
    cmTechnologyVideoCamera = 'vidc',
    cmTechnologyProjectionTelevision = 'pjtv',
    cmTechnologyCRTDisplay = 'CRT ',
    cmTechnologyPMDisplay = 'PMD ',
    cmTechnologyAMDisplay = 'AMD ',
    cmTechnologyPhotoCD = 'KPCD',
    cmTechnologyPhotoImageSetter = 'imgs',
    cmTechnologyGravure = 'grav',
    cmTechnologyOffsetLithography = 'offs',
    cmTechnologySilkscreen = 'silk',
    cmTechnologyFlexography = 'flex'
};
```

Constants

`cmTechnologyDigitalCamera`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyFilmScanner`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyReflectiveScanner`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyInkJetPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyThermalWaxPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyElectrophotographicPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyElectrostaticPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyDyeSublimationPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPhotographicPaperPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyFilmWriter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyVideoMonitor`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyVideoCamera`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyProjectionTelevision`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyCRTDisplay`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPMDisplay`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyAMDisplay`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPhotoCD`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPhotoImageSetter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyGravure`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyOffsetLithography`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologySilkscreen`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyFlexography`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Use Types

Specify use types.

```
enum {
    cmInputUse = 'inpt',
    cmOutputUse = 'outp',
    cmDisplayUse = 'dply',
    cmProofUse = 'pruf'
};
```

Constants

`cmInputUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmOutputUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmDisplayUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmProofUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion
Used for the function `CMGetProfileByUse` and `SetDefaultProfileByUse`.

Video Card Gamma Storage Types

Specify data storage type constants.

```
enum {
    cmVideoCardGammaTableType = 0,
    cmVideoCardGammaFormulaType = 1
};
```

Constants

`cmVideoCardGammaTableType`
The video card gamma data is stored in a table format. See `CMVideoCardGammaTable` (page 182) for a description of the table format.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmVideoCardGammaFormulaType`
The video card gamma tag data is stored as a formula. See `CMVideoCardGammaFormula` (page 181) for a description of the formula format.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Discussion
A video card gamma profile tag can store gamma data either as a formula or as a table of values. You use a storage type constant to specify which data storage type the tag uses.

If the video card uses a different format than the format you specify (for example, the card uses data in table format and you supply data in formula format), ColorSync will adapt the data you supply to match the format the card expects.

Version Notes
Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. As of version 2.5, the only ColorSync function that attempts to take advantage of video card gamma data is `CMSetProfileByAVID` (page 72).

Video Card Gamma Tags

Specify video card gamma information.

```
enum {
    cmPS2CRDVMSizeTag = 'psvm',
    cmVideoCardGammaTag = 'vcgt',
    cmMakeAndModelTag = 'mmod',
    cmProfileDescriptionMLTag = 'dscm',
    cmNativeDisplayInfoTag = 'ndin'
};
```

Constants

`cmPS2CRDVMSizeTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmVideoCardGammaTag`
Constant for profile tag that specifies video card gamma information. When you create a tag to store video card gamma data in a profile, you use the `cmVideoCardGammaTag` constant to specify the tag.
Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. As of version 2.5, the only ColorSync function that attempts to take advantage of video card gamma data is `CMSetProfileByAVID` (page 72).
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMakeAndModelTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmProfileDescriptionMLTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNativeDisplayInfoTag`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

Video Card Gamma Signatures

Specify signatures used for video card gamma information.

```
enum {
    cmSigPS2CRDVMSizeType = 'psvm',
    cmSigVideoCardGammaType = 'vcgt',
    cmSigMakeAndModelType = 'mmod',
    cmSigNativeDisplayInfoType = 'ndin',
    cmSigMultiLocalizedUniCodeType = 'mluc'
};
```

Constants

`cmSigPS2CRDVMSizeType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigVideoCardGammaType`

Constant that specifies video card gamma type signature in a video card gamma profile tag. That is, you use this constant to set the `typeDescriptor` field of the `CMVideoCardGammaType` (page 182) structure. There is currently only one type possible for a video card gamma tag.

Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. As of version 2.5, the only ColorSync function that attempts to take advantage of video card gamma data is `CMSetProfileByAVID` (page 72).

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigMakeAndModelType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigNativeDisplayInfoType`

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmSigMultiLocalizedUniCodeType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Result Codes

The most common result codes returned by ColorSync Manager are listed below.

Result Code	Value	Description
`noErr`	0	No error Available in Mac OS X v10.0 and later.
`cmProfileError`	-170	There is something wrong with the content of the profile Available in Mac OS X v10.0 and later.

Result Code	Value	Description
cmMethodError	-171	An error occurred during the CMM arbitration process that determines the CMM to use Available in Mac OS X v10.0 and later.
cmMethodNotFound	-175	CMM not present Available in Mac OS X v10.0 and later.
cmProfileNotFound	-176	Responder error Available in Mac OS X v10.0 and later.
cmProfilesIdentical	-177	Profiles are the same Available in Mac OS X v10.0 and later.
cmCantConcatenateError	-178	Profiles cannot be concatenated Available in Mac OS X v10.0 and later.
cmCantXYZ	-179	CMM does not handle XYZ color space Available in Mac OS X v10.0 and later.
cmCantDeleteProfile	-180	Responder error Available in Mac OS X v10.0 and later.
cmUnsupportedDataType	-181	Responder error Available in Mac OS X v10.0 and later.
cmNoCurrentProfile	-182	Responder error Available in Mac OS X v10.0 and later.
cmElementTagNotFound	-4200	The tag you specified is not in the specified profile Available in Mac OS X v10.0 and later.
cmIndexRangeErr	-4201	Tag index out of range Available in Mac OS X v10.0 and later.
cmCantDeleteElement	-4202	Cannot delete the specified profile element Available in Mac OS X v10.0 and later.
cmFatalProfileErr	-4203	Returned from File Manager while updating a profile file in response to CMUpdateProfile; profile content may be corrupted Available in Mac OS X v10.0 and later.
cmInvalidProfile	-4204	Profile reference is invalid or refers to an inappropriate profile Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`cmInvalidProfileLocation`	-4205	Operation not supported for this profile location Available in Mac OS X v10.0 and later.
`cmInvalidSearch`	-4206	Bad search handle Available in Mac OS X v10.0 and later.
`cmSearchError`	-4207	Internal error occurred during profile search Available in Mac OS X v10.0 and later.
`cmErrIncompatibleProfile`	-4208	Unspecified profile error Available in Mac OS X v10.0 and later.
`cmInvalidColorSpace`	-4209	Profile color space does not match bitmap type Available in Mac OS X v10.0 and later.
`cmInvalidSrcMap`	-4210	Source pixel map or bitmap was invalid Available in Mac OS X v10.0 and later.
`cmInvalidDstMap`	-4211	Destination pix/bit map was invalid Available in Mac OS X v10.0 and later.
`cmNoGDevicesError`	-4212	Begin matching or end matching—no graphics devices available Available in Mac OS X v10.0 and later.
`cmInvalidProfileComment`	-4213	Bad profile comment during `drawpicture` Available in Mac OS X v10.0 and later.
`cmRangeOverFlow`	-4214	One or more output color value overflows in color conversion; all input color values will be converted and the overflow will be clipped Available in Mac OS X v10.0 and later.
`cmCantCopyModifiedV1Profile`	-4215	It is illegal to copy version 1.0 profiles that have been modified Available in Mac OS X v10.0 and later.
`cmNamedColorNotFound`	-4216	The specified named color was not found in the specified profile Available in Mac OS X v10.0 and later.
`cmCantGamutCheckError`	-4217	Gamut checking not supported by this color world—that is, the color world does not contain a gamut table because it was built with gamut checking turned off Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`cmDeviceDBNotFoundErr`	-4227	Preferences not found or loaded; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
`cmDeviceAlreadyRegistered`	-4228	Device already registered; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
`cmDeviceNotRegistered`	-4229	Device not found; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
`cmDeviceProfilesNotFound`	-4230	Profiles not found; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
`cmInternalCFErr`	-4231	CoreFoundation failure; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.

Deprecated ColorSync Manager Functions

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

Deprecated in Mac OS X v10.4

CMEnableMatchingComment

Inserts a comment into the currently open picture to turn matching on or off. (Deprecated in Mac OS X v10.4.)

```
void CMEnableMatchingComment (
   Boolean enableIt
);
```

Parameters

enableIt

A flag that directs the ColorSync Manager to generate a `cmEnableMatchingPicComment` comment if `true`, or a `cmDisbleMatchingPicComment` comment if `false`.

Discussion

If you call this function when no picture is open, it will have no effect.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CMEndMatching

Concludes a QuickDraw-specific ColorSync matching session initiated by a previous call to the `NCMBeginMatching` function. (Deprecated in Mac OS X v10.4.)

```
void CMEndMatching (
   CMMatchRef myRef
);
```

Parameters

myRef

A reference to the matching session to end. This reference was previously created and returned by a call to `NCMBeginMatching` function. See the QuickDraw Reference for a description of the PixMap data type.

Discussion
The `CMEndMatching` function releases private memory allocated for the QuickDraw-specific matching session.

After you call the `NCMBeginMatching` function and before you call `CMEndMatching` to end the matching session, embedded color-matching picture comments, such as `cmEnableMatching` and `cmDisableMatching`, are not acknowledged.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CWCheckPixMap

Checks the colors of a pixel map using the profiles of a specified color world to determine whether the colors are in the gamut of the destination device. (Deprecated in Mac OS X v10.4.)

```
CMError CWCheckPixMap (
   CMWorldRef cw,
   PixMap *myPixMap,
   CMBitmapCallBackUPP progressProc,
   void *refCon,
   BitMap *resultBitMap
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 183) in which color checking is to occur.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both return color world references of type `CMWorldRef` (page 183).

See the QuickDraw Reference for a description of the PixMap data type.

myPixMap

A pointer to the pixel map to check colors for. A pixel map is a QuickDraw structure describing pixel data. The pixel map must be nonrelocatable; to ensure this, you should lock the handle to the pixel map. See the QuickDraw Reference for a description of the PixMap data type.

progressProc

A calling program–supplied callback function that allows your application to monitor progress or abort the operation as the pixel map colors are checked against the gamut of the destination device.

The default CMM calls your function approximately every half-second unless color checking occurs in less time; this happens when there is a small amount of data to be checked. If the function returns a result of `true`, the operation is aborted. Specify `NULL` for this parameter if your application will not monitor the pixel map color checking. For information on the callback function and its type definition, see the function `CMBitmapCallBackProcPtr` (page 93).

See the QuickDraw Reference for a description of the PixMap data type.

refCon

A pointer to a reference constant for application data passed as a parameter to calls to your `CMBitmapCallBack` function pointed to by `progressProc`.

resultBitMap

A pointer to a QuickDraw bitmap. On return, bits are set to 1 if the corresponding pixel of the pixel map indicated by `myPixMap` is out of gamut. Boundaries of the bitmap indicated by `resultBitMap` must equal the parameter of the pixel map indicated by the `myPixMap`. See the QuickDraw Reference for a description of the PixMap data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CWCheckPixMap` function performs a gamut test of the pixel data of the `myPixMap` pixel map to determine if its colors are within the gamut of the destination device as specified by the destination profile. The gamut test provides a preview of color matching using the specified color world.

The preferred CMM, as determined by the ColorSync Manager based on the profiles of the color world configuration, is called to perform the color matching.

If the preferred CMM is not available, then the ColorSync Manager calls the default CMM to perform the matching. If the preferred CMM is available but does not implement the `CMCheckPixmap` function, then the ColorSync Manager unpacks the colors in the pixel map to create a color list and calls the preferred CMM's `CMCheckColors` function, passing to this function the list of colors to match. Every CMM must support the `CMCheckColors` function.

For this function to execute successfully, the source and destination profiles' data color spaces (`dataColorSpace` field) must be RGB to match the data color space of the pixel map, which is implicitly RGB.

If you specify a pointer to a callback function in the `progressProc` parameter, the CMM performing the color checking calls your function to monitor progress of the session. Each time the CMM calls your function, it passes the function any data you specified in the `CWCheckPixMap` function's `refCon` parameter.

You can use the reference constant to pass in any kind of data your callback function requires. For example, if your application uses a dialog box with a progress bar to inform the user of the color-checking session's progress, you can use the reference constant to pass the dialog box's window reference to the callback routine. For information about the callback function, see the function `CMBitmapCallBackProcPtr` (page 93).

You should ensure that the buffer pointed to by the `baseAddr` field of the bitmap passed in the `resultBitMap` parameter is zeroed out.

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CWMatchPixMap

Matches a pixel map in place based on a specified color world. (Deprecated in Mac OS X v10.4.)

```
CMError CWMatchPixMap (
   CMWorldRef cw,
   PixMap *myPixMap,
   CMBitmapCallBackUPP progressProc,
   void *refCon
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 183) in which matching is to occur.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

myPixMap

A pointer to the pixel map to match. A pixel map is a QuickDraw structure describing pixel data. The pixel map must be nonrelocatable; to ensure this, you should lock the handle to the pixel map before you call this function. See the QuickDraw Reference for a description of the `PixMap` data type.

progressProc

A function supplied by your application to monitor progress or abort the operation as the pixel map colors are matched. The default CMM calls your function approximately every half-second, unless matching is completed in less time.

If the function returns a result of `true`, the operation is aborted. You specify `NULL` for this parameter if your application will not monitor the pixel map color matching. For information on the callback function and its type definition, refer to the function `CMProfileFilterProcPtr` (page 105).

refCon

A pointer to a reference constant for application data that is passed as a parameter to calls to `progressProc`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CWMatchPixMap` function matches a pixel map in place using the profiles specified by the given color world. The preferred CMM, as determined by the ColorSync Manager based on the color world configuration, is called to perform the color matching.

If the preferred CMM is not available, then the ColorSync Manager calls the default CMM to perform the matching. If the preferred CMM is available but it does not implement the `CMMatchPixMap` function, then the ColorSync Manager unpacks the colors in the pixel map to create a color list and calls the preferred CMM's `CMMatchColors` function, passing to this function the list of colors to match. Every CMM must support the `CMMatchColors` function.

For this function to execute successfully, the source and destination profiles' data color spaces (`dataColorSpace` field) must be RGB to match the data color space of the pixel map, which is implicitly RGB. For color spaces other than RGB, you should use the function `CWMatchBitmap` (page 86).

If you specify a pointer to a callback function in the `progressProc` parameter, the CMM performing the color matching calls your function to monitor progress of the session. Each time the CMM calls your function, it passes the function any data you specified in the `CWMatchPixMap` function's `refCon` parameter. If the ColorSync Manager performs the color matching, it calls your callback monitoring function once every scan line during this process.

You can use the reference constant to pass in any kind of data your callback function requires. For example, if your application uses a dialog box with a progress bar to inform the user of the color-matching session's progress, you can use the reference constant to pass the dialog box's window reference to the callback routine. For information about the callback function, see the function `CMBitmapCallBackProcPtr` (page 93).

Applications do not interact directly with the function `CWMatchColors` (page 87).

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NCMBeginMatching

Sets up a QuickDraw-specific ColorSync matching session, using the specified source and destination profiles. (Deprecated in Mac OS X v10.4.)

```
CMError NCMBeginMatching (
   CMProfileRef src,
   CMProfileRef dst,
   CMMatchRef *myRef
);
```

Parameters

src

A profile reference of type `CMProfileRef` (page 166) that specifies the source profile for the matching session. Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 33) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 44) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

See the QuickDraw Reference for a description of the PixMap data type.

dst

A profile reference of type `CMProfileRef` (page 166) that specifies the destination profile for the matching session. Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 33) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 44) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.See the QuickDraw Reference for a description of the PixMap data type.

myRef

A pointer to a matching session. On return, it specifies the QuickDraw-specific matching session that was set up. See the QuickDraw Reference for a description of the PixMap data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `NCMBeginMatching` function sets up a QuickDraw-specific matching session, telling the ColorSync Manager to match all colors drawn to the current graphics device using the specified source and destination profiles.

The `NCMBeginMatching` function returns a reference to the color-matching session. You must later pass this reference to the function `CMEndMatching` (page 265) to conclude the session.

The source and destination profiles define how the match is to occur. Passing `NULL` for either the source or destination profile is equivalent to passing the system profile. If the current device is a screen device, matching to all screen devices occurs.

The `NCMBeginMatching` and `CMEndMatching` functions can be nested. In such cases, the ColorSync Manager matches to the most recently added profiles first. Therefore, if you want to use the `NCMBeginMatching`–`CMEndMatching` pair to perform a page preview—which typically entails color matching from a source device (scanner) to a destination device (printer) to a preview device (display)— you first call `NCMBeginMatching` with the printer-to-display profiles, and then call `NCMBeginMatching` with the scanner-to-printer profiles. The ColorSync Manager then matches all drawing from the scanner to the printer and then back to the display. The print preview process entails multiprofile transformations. The ColorSync Manager general purpose functions (which include the use of concatenated profiles well suited to print-preview processing) offer an easier and faster way to do this. These functions are described in "Matching Colors Using General Purpose Functions".

If you call `NCMBeginMatching` before drawing to the screen's graphics device (as opposed to an offscreen device), you must call `CMEndMatching` to finish a matching session before calling `WaitNextEvent` or any other routine (such as Window Manager routines) that could draw to the screen. Failing to do so will cause unwanted matching to occur. Furthermore, if a device has color matching enabled, you cannot call the `CopyBits` procedure to copy from it to itself unless the source and destination rectangles are the same.

Even if you call the `NCMBeginMatching` function before calling the QuickDraw `DrawPicture` function, the ColorSync picture comments such as `cmEnableMatching` and `cmDisableMatching` are not acknowledged. For the ColorSync Manager to recognize these comments and allow their use, you must call the function `NCMUseProfileComment` (page 272) for color matching using picture comments.

This function causes matching for the specified devices rather than for the current color graphics port.

The `NCMBeginMatching` function uses QuickDraw and performs color matching in a manner acceptable to most applications. However, if your application needs a finer level of control over color matching, it can use the general purpose functions described in "Matching Colors Using General Purpose Functions".

Version Notes
The parameter descriptions for `src` and `dst` describe changes in how this function is used starting with ColorSync version 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NCMDrawMatchedPicture

Matches a picture's colors to a destination device's color gamut, as the picture is drawn, using the specified destination profile. (Deprecated in Mac OS X v10.4.)

```
void NCMDrawMatchedPicture (
   PicHandle myPicture,
   CMProfileRef dst,
   Rect *myRect
);
```

Parameters

myPicture

The QuickDraw picture whose colors are to be matched. See the QuickDraw Reference for a description of the PixMap data type.

dst

A profile reference of type `CMProfileRef` (page 166) to the profile of the destination device. Starting with ColorSync version 2.5, if you know the destination display device, you can call `CMGetProfileByAVID` (page 44) to get the specific profile for the display, or you can call `CMGetDefaultProfileBySpace` (page 33) to get the default profile for the RGB color space,.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

See the QuickDraw Reference for a description of the PixMap data type.

myRect

A pointer to a destination rectangle for rendering the picture specified by `myPicture`.

Return Value

This function does not return an error value. Instead, after calling `NCMDrawMatchedPicture` you call the `QDError` routine to determine if an error has occurred.

Discussion

The `NCMDrawMatchedPicture` function operates in the context of the current color graphics port. This function sets up and takes down a color-matching session. It automatically matches all colors in a picture to the destination profile for a destination device as the picture is drawn. It uses the ColorSync system profile as the initial source profile and any embedded profiles as they are encountered thereafter. (Because color-matching picture comments embedded in the picture to be matched are recognized, embedded profiles are used.)

The ColorSync Manager defines five picture comment kinds, as described in "Picture Comment Kinds" (page 236). For embedding to work correctly, each embedded profile that is used for matching must be terminated by a picture comment of kind `cmEndProfile`. If a picture comment is not specified to end the profile after drawing operations using that profile are performed, the profile will remain in effect until another embedded profile is introduced that has a picture comment kind of `cmBeginProfile`. To avoid unexpected matching effects, always pair use of the `cmBeginProfile` and `cmEndProfile` picture comments. When the ColorSync Manager encounters a `cmEndProfile` picture comment, it restores use of the system profile for matching until it encounters another `cmBeginProfile` picture comment.

The picture is drawn with matched colors to all screen graphics devices. If the current graphics device is not a screen device, matching occurs for that graphics device only.

If the current port is not a color graphics port, then calling this function is equivalent to calling `DrawPicture`, in which case no color matching occurs.

Version Notes
The parameter description for `dst` describes changes in how this function is used starting with ColorSync version 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NCMUseProfileComment

Automatically embeds a profile or a profile identifier into an open picture. (Deprecated in Mac OS X v10.4.)

```
CMError NCMUseProfileComment (
   CMProfileRef prof,
   UInt32 flags
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile to embed. See the QuickDraw Reference for a description of the PixMap data type.

flags

A flag value in which individual bits determine settings. "Embedded Profile Identifiers" (page 223) describes constants for use with this parameter. For example, you pass `cmEmbedWholeProfile` to embed a whole profile or `cmEmbedProfileIdentifier` to embed a profile identifier. No other values are currently defined; all other bits are reserved for future use.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `NCMUseProfileComment` function automatically generates the picture comments required to embed the specified profile or profile identifier into the open picture.

To embed a profile, you use the constant `cmEmbedWholeProfile` to set the `flags` parameter before calling `NCMUseProfileComment`. The `NCMUseProfileComment` function calls the QuickDraw `PicComment` function with a picture comment `kind` value of `cmComment` and a 4-byte selector that describes the type of data in the picture comment: `cmBeginProfileSel` to begin the profile, `cmContinueProfileSel` to continue, and `cmEndProfileSel` to end the profile. These constants are described in "Picture Comment Selectors" (page 238).

If the size in bytes of the profile and the 4-byte selector together exceed 32 KB, this function segments the profile data and embeds the multiple segments in consecutive order using selector `cmContinueProfileSel` to embed each segment.

To embed a profile identifier of type `CMProfileIdentifier` (page 162) , you use the constant `cmEmbedProfileIdentifier` to set the `flags` parameter before calling `NCMUseProfileComment`. The function extracts the necessary information from the profile reference (`prof`) to embed a profile identifier for the profile. The profile reference can refer to a previously embedded profile, or to a profile on disk in the ColorSync Profiles folder.

You can use this function to embed most types of profiles in an image, including device link profiles, but not abstract profiles. You cannot use this function to embed ColorSync 1.0 profiles in an image.

The `NCMUseProfileComment` function precedes the profile it embeds with a picture comment of kind `cmBeginProfile`. For embedding to work correctly, the currently effective profile must be terminated by a picture comment of kind `cmEndProfile` after drawing operations using that profile are performed. You are responsible for adding the picture comment of kind `cmEndProfile`. If a picture comment was not specified to end the profile following the drawing operations to which the profile applies, the profile will remain in effect until the next embedded profile is introduced with a picture comment of kind `cmBeginProfile`. However, use of the next profile might not be the intended action. Always pair use of the `cmBeginProfile` and `cmEndProfile` picture comments. When the ColorSync Manager encounters a `cmEndProfile` picture comment, it restores use of the system profile for matching until it encounters another `cmBeginProfile` picture comment.

Version Notes
In ColorSync 2.0, the `flags` parameter was ignored and the routine always embedded the entire profile.

In ColorSync 2.0, if the `prof` parameter refers to a version 1.0 profile, the profile is not embedded into the picture correctly. In ColorSync versions starting with 2.1, this bug has been fixed. One possible workaround for this problem in ColorSync 2.0 is to call `CMCopyProfile` to copy the 1.0 profile reference into a handle. The handle can then be embedded into the picture using `CMUseProfileComment`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

Deprecated in Mac OS X v10.5

CMConvertFixedXYZToXYZ

Converts colors specified in XYZ color space whose components are expressed as Fixed XYZ 32-bit signed values of type `CMFixedXYZColor` to equivalent colors expressed as XYZ 16-bit unsigned values of type `CMXYZColor`. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertFixedXYZToXYZ (
   const CMFixedXYZColor *src,
   CMXYZColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of Fixed XYZ colors to convert to XYZ colors.

dst

A pointer to an array containing the list of colors resulting from the conversion specified as XYZ colors.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMConvertFixedXYZToXYZ` function converts one or more colors defined in the Fixed XYZ color space to equivalent colors defined in the XYZ color space. The XYZ color space is device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertFixedXYZToXYZ` function to overwrite the source colors with the resulting converted color specifications.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertHLSToRGB

Converts colors specified in the HLS color space to equivalent colors defined in the RGB color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertHLSToRGB (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of HLS colors to convert to RGB colors.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the RGB color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertHLSToRGB` function converts one or more colors defined in the HLS color space to equivalent colors defined in the RGB color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertHLSToRGB` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertHSVToRGB

Converts colors specified in the HSV color space to equivalent colors defined in the RGB color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertHSVToRGB (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of HSV colors to convert to RGB colors.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the RGB color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertHSVToRGB` function converts one or more colors defined in the HSV color space to equivalent colors defined in the RGB color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertHSVToRGB` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertLabToXYZ

Converts colors specified in the L*a*b* color space to the XYZ color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertLabToXYZ (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to a buffer containing the list of L*a*b* colors to convert to XYZ colors.

white
A pointer to a reference white point.

dst
A pointer to a buffer containing the list of colors as specified in the XYZ color space resulting from the conversion.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertLabToXYZ` function converts one or more colors defined in the L*a*b color space to equivalent colors defined in the XYZ color space. Both color spaces are device independent.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertLuvToXYZ

Converts colors specified in the L*u*v* color space to the XYZ color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertLuvToXYZ (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of L*u*v* colors to convert.

white

A pointer to a reference white point.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the XYZ color space.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMConvertLuvToXYZ` function converts one or more colors defined in the L*u*v color space to equivalent colors defined in the XYZ color space. Both color spaces are device independent.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertRGBToGray

Converts colors specified in the RGB color space to equivalent colors defined in the Gray color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertRGBToGray (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of colors specified in RGB space to convert to colors specified in Gray space.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the Gray color space.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertRGBToGray` function converts one or more colors defined in the RGB color space to equivalent colors defined in the Gray color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertRGBToGray` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertRGBToHLS

Converts colors specified in the RGB color space to equivalent colors defined in the HLS color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertRGBToHLS (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to an array containing the list of RGB colors to convert to HLS colors.

dst
A pointer to an array containing the list of colors, resulting from the conversion, as specified in the HLS color space.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertRGBToHLS` function converts one or more colors defined in the RGB color space to equivalent colors defined in the HLS color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertRGBToHLS` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertRGBToHSV

Converts colors specified in the RGB color space to equivalent colors defined in the HSV color space when the device types are the same. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertRGBToHSV (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to an array containing the list of RGB colors to convert to HSV colors.

dst
A pointer to an array containing the list of colors, resulting from the conversion, as specified in the HSV color space.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertRGBToHSV` function converts one or more colors defined in the RGB color space to equivalent colors defined in the HSV color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertRGBToHSV` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToFixedXYZ

Converts colors specified in the XYZ color space whose components are expressed as XYZ 16-bit unsigned values of type `CMXYZColor` to equivalent colors expressed as 32-bit signed values of type `CMFixedXYZColor`. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToFixedXYZ (
   const CMXYZColor *src,
   CMFixedXYZColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of XYZ colors to convert to Fixed XYZ colors.

dst

A pointer to an array containing the list of colors resulting from the conversion in which the colors are specified as Fixed XYZ colors.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMConvertXYZToFixedXYZ` function converts one or more colors whose components are defined as XYZ colors to equivalent colors whose components are defined as Fixed XYZ colors. Fixed XYZ colors allow for 32-bit precision. The XYZ color space is device independent.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertXYZToLab

Converts colors specified in the XYZ color space to the L*a*b* color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToLab (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of XYZ colors to convert to L*a*b* colors.

white
A pointer to a reference white point.

dst
A pointer to an array containing the list of L*a*b* colors resulting from the conversion.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertXYZToLab` function converts one or more colors defined in the XYZ color space to equivalent colors defined in the L*a*b* color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertXYZToLab` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToLuv

Converts colors specified in the XYZ color space to the L*u*v* color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToLuv (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to an array containing the list of XYZ colors to convert to L*u*v* colors.

white
A pointer to a reference white point.

dst
A pointer to an array containing the list of colors represented in L*u*v* color space resulting from the conversion.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertXYZToLuv` function converts one or more colors defined in the XYZ color space to equivalent colors defined in the L*u*v* color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertXYZToLuv` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToXYZ

Converts a source color to a destination color using the specified chromatic adaptation method. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToXYZ (
   const CMColor *src,
   const CMXYZColor *srcIlluminant,
   CMColor *dst,
   const CMXYZColor *dstIlluminant,
   CMChromaticAdaptation method,
   size_t count
);
```

Parameters
src
srcIlluminant
dst
dstIlluminant
method
count

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToYxy

Converts colors specified in the XYZ color space to the Yxy color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToYxy (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of XYZ colors to convert to Yxy colors.

dst

A pointer to an array containing the list of colors resulting from the conversion represented in the Yxy color space.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMConvertXYZToYxy` function converts one or more colors defined in the XYZ color space to equivalent colors defined in the Yxy color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertXYZToYxy` function to overwrite the source colors with the resulting converted color specifications.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertYxyToXYZ

Converts colors specified in the Yxy color space to the XYZ color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertYxyToXYZ (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of Yxy colors to convert.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the XYZ color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The `CMConvertYxyToXYZ` function converts one or more colors defined in the Yxy color space to equivalent colors defined in the XYZ color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertYxyToXYZ` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMCountImageProfiles

Obtains a count of the number of embeded profiles for a given image. (Deprecated in Mac OS X v10.5.)

```
CMError CMCountImageProfiles (
   const FSSpec *spec,
   UInt32 *count
);
```

Parameters

spec

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

count

On output, a count of the embeded profiles for the image

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMCreateProfileIdentifier

Creates a profile identifier for a specified profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMCreateProfileIdentifier (
   CMProfileRef prof,
   CMProfileIdentifierPtr ident,
   UInt32 *size
);
```

Parameters

prof

ident

size

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMDisposeProfileSearch

Frees the private memory allocated for a profile search after your application has completed the search. (Deprecated in Mac OS X v10.5.)

```
void CMDisposeProfileSearch (
   CMProfileSearchRef search
);
```

Parameters

search

A reference to the profile search result list whose private memory is to be released. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 168). See the QuickDraw Reference for a description of the PixMap data type.

Discussion

To set up a search, use the function `CMNewProfileSearch` (page 296). To obtain a reference to a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfile` (page 302). To obtain the file specification for a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfileFileSpec` (page 302). To update the search result list, use the function `CMUpdateProfileSearch` (page 308).

Version Notes

This function is not recommended for use in ColorSync 2.5.

Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 57) for profile searching.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMEmbedImage

Embeds an image with an ICC profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMEmbedImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   CMProfileRef embProf
);
```

Parameters

specFrom
A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto
If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl
A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

embProf
The profile to embed in the image.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMFlattenProfile

Transfers a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document. (Deprecated in Mac OS X v10.5.)

```
CMError CMFlattenProfile (
   CMProfileRef prof,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile to flatten.

flags

Reserved for future use.

proc

A pointer to a function that you provide to perform the low-level data transfer. For more information, see the function `CMFlattenProcPtr` (page 96).

refCon

A pointer to a reference constant for application data which the color management module (CMM) passes to the `CMFlattenProcPtr` function each time it calls the function. For example, the reference constant may point to a data structure that holds information required by the `CMFlattenProcPtr` function to perform the data transfer, such as the reference number to a disk file in which the flattened profile is to be stored.

Starting with ColorSync version 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM.

preferredCMMnotfound

A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM specified by the profile was not available to perform flattening or does not support this function and the default CMM was used. Has the value `false` if the profile's preferred CMM is able to perform flattening.

Starting with ColorSync 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM. On return, the value of `preferredCMMnotfound` is guaranteed to be `false`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The ColorSync Manager passes to the CMM the pointer to your profile-flattening function. The CMM calls your function `CMFlattenProcPtr` (page 96) to perform the actual data transfer.

To unflatten a profile embedded in a graphics document to an independent disk file, use the function "Accessing Profile Elements".

Version Notes

Prior to version 2.5, the ColorSync Manager dispatches the `CMFlattenProfile` function to the CMM specified by the profile whose reference you provide. If the preferred CMM is unavailable or it does not support this function, then the default CMM is used.

Starting with ColorSync version 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM. As a result, the value returned in the `preferredCMMnotfound` parameter is guaranteed to be `false`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetColorSyncFolderSpec

Obtains the volume reference number and the directory ID for a ColorSync Profiles folder. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetColorSyncFolderSpec (
   short vRefNum,
   Boolean createFolder,
   short *foundVRefNum,
   long *foundDirID
);
```

Parameters

vRefNum

The location of the ColorSync profiles folder. In Mac OS X, pass a constant that specifies one of the four possible locations for ColorSync profiles. Pass `kSystemDomain` for profiles located in:

`/System/Library/ColorSync/Profiles`

Pass `kLocalDomain` for profiles located in:

`/Library/ColorSync/Profiles`

Pass `kNetworkDomain` for profiles located in:

`/Network/Library/ColorSync/Profiles`

Pass `kUserDomain` for profiles located in:

`~/Library/ColorSync/Profiles`

In Mac OS 9, pass the reference number of the volume to examine. The volume must be mounted. The constant `kOnSystemDisk` defined in the Folders header file (`Folders.h`) specifies the active system volume.

createFolder

A flag you set to `true` to direct the ColorSync Manager to create the ColorSync Profiles folder, if it does not exist. You can use the constants `kCreateFolder` and `kDontCreateFolder`, defined in the `Folders.h` header file, to assign a value to the flag.

foundVRefNum

A pointer to a volume reference number. On return, the volume reference number for the volume on which the ColorSync Profiles folder resides.

foundDirID

A pointer to a directory ID. On return, the directory ID for the volume on which the ColorSync Profiles folder resides.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
If the ColorSync Profiles folder does not already exist, you can use this function to create it.

Version Notes
Starting with version 2.5, the name and location of the profile folder changed.

Your application should use the function `CMIterateColorSyncFolder` (page 57), available starting in ColorSync version 2.5, or one of the search functions described in "Searching for Profiles Prior to ColorSync 2.5", to search for a profile file, even if it is only looking for one file. Do not search for a profile file by obtaining the location of the profiles folder and searching for the file directly.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetCWInfo

Obtains information about the color management modules (CMMs) used for a specific color world. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetCWInfo (
   CMWorldRef cw,
   CMCWInfoRecord *info
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 183) about which you want information.

The functions `NCWNewColorWorld` (page 90) and `CWConcatColorWorld` (page 83) both allocate color world references of type `CMWorldRef` (page 183).

info

A pointer to a color world information record of type `CMCWInfoRecord` (page 129) that your application supplies. On return, the ColorSync Manager returns information in this structure describing the number of CMMs involved in the matching session and the CMM type and version of each CMM used.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
This discussion is accurate for versions of ColorSync prior to 2.5. See the version notes below for changes starting with version 2.5.

To learn whether one or two CMMs are used for color matching and color checking in a given color world and to obtain the CMM type and version number of each CMM used, your application must first obtain a reference to the color world. To obtain a reference to a ColorSync color world, you (or some other process) must have created the color world using the function `NCWNewColorWorld` (page 90) or the function `CWConcatColorWorld` (page 83).

The source and destination profiles you specify when you create a color world identify their preferred CMMs, and you explicitly identify the profile whose CMM is used for a device link profile or a concatenated color world. However, you cannot be certain if the specified CMM will actually be used until the ColorSync Manager determines internally if the CMM is available and able to perform the requested function. For example, when the specified CMM is not available, the default CMM is used.

The `CMGetCWInfo` function identifies the CMM or CMMs to use. Your application must allocate a data structure of type `CMCWInfoRecord` and pass a pointer to it in the `info` parameter. The `CMGetCWInfo` function returns the color world information in this structure. The structure includes a `cmmCount` field identifying the number of CMMs that will be used and an array of two members containing structures of type `CMMInfoRecord` (page 150). The `CMGetCWInfo` function returns information in one or both of the CMM information records depending on whether one or two CMMs are used.

Version Notes
Starting with ColorSync 2.5, a user can select a preferred CMM with the ColorSync control panel. If the user has selected a preferred CMM, and if it is available, then it will be used for all color conversion and matching operations.

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetDeviceProfiles

Gets the profiles used by a given device. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetDeviceProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   UInt32 *arraySize,
   CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass
The device class for the device whose profiles you want to get. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID
The device ID for the device whose profiles you want to get.

arraySize
A pointer to the size of the array to be returned. You can first call this routine to get the size returned, then call it again with the size of the buffer to receive the array.

deviceProfiles
On output, an array of profiles used by the device. You can first pass `NULL` in this parameter to receive the size of the array in the `arraySize` parameter. Then, once the appropriate amount of storage has been allocated, a pointer to it can be passed in this parameter to have the array copied to that storage.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMDeviceIntegration.h`

CMGetImageSpace

Returns the signature of the data color space in which the color values of colors in an image are expressed. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetImageSpace (
   const FSSpec *spec,
   OSType *space
);
```

Parameters

spec

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

space

The signature of the data color space of the color values of colors for the image file is returned here.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMGetIndImageProfile

Obtains a specific embeded profile for a given image. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetIndImageProfile (
   const FSSpec *spec,
   UInt32 index,
   CMProfileRef *prof
);
```

Parameters

spec

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

index

The numeric index of the profile to return.

prof

On output, points to the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMGetPreferredCMM

Identifies the preferred CMM specified by the ColorSync control panel. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetPreferredCMM (
   OSType *cmmType,
   Boolean *prefCMMnotfound
);
```

Parameters

cmmType

A pointer to an `OSType`. On return, the component subtype for the preferred CMM. For example, the subtype for ColorSync's default CMM is `'appl'` and the subtype for the Kodak CMM is `'KCMS'`. A return value of `NULL` indicates the preferred CMM in the ColorSync control panel is set to Automatic.

preferredCMMnotfound

A pointer to a Boolean flag for whether the preferred CMM was not found. On return, has the value `true` if the CMM was not found, `false` if it was found.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMGetPreferredCMM` function returns in the `cmmType` parameter a value that identifies the preferred CMM the user last specified in the ColorSync control panel. `CMGetPreferredCMM` returns `false` in the `preferredCMMnotfound` parameter if the preferred CMM is currently available and `true` if it is not. The

preferred CMM may not be available, for example, because a user specifies a preferred CMM in the ColorSync control panel, then reboots with extensions off. ColorSync does not change the preferred CMM setting when the preferred CMM is not available.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetProfileLocation

Obtains the location of a profile based on the specified profile reference. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetProfileLocation (
   CMProfileRef prof,
   CMProfileLocation *location
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166). Before calling `CMGetProfileLocation`, you set the reference to specify the profile you wish to obtain the location for.

theProfile

A pointer to a profile location structure of type `CMProfileLocation` (page 165). On return, specifies the location of the profile. Commonly, a profile is disk-file based, but it may instead be temporary, handle-based, pointer-based, or accessed through a procedure supplied by your application.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
When your application calls the `CMValidateProfile` function, the ColorSync Manager dispatches the function to the CMM specified by the `CMMType` header field of the profile whose reference you specify. The preferred CMM can support this function or not.

To open a profile and obtain a reference to it, use the function `CMOpenProfile` (page 63).

Version Notes
This function is not recommended for use in ColorSync 2.5.

Starting with ColorSync version 2.5, you should use the function `NCMGetProfileLocation` (page 88) instead of `CMGetProfileLocation`.

As of version 2.5, if you call `CMGetProfileLocation`, it will just call `NCMGetProfileLocation` in turn, passing the profile specified by `prof`, the profile location specified by `theProfile`, and a location size value of `cmOriginalProfileLocationSize`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

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

Declared In
CMApplication.h

CMGetScriptProfileDescription

Obtains the internal name (or description) of a profile and the script code identifying the language in which the profile name is specified from the specified profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetScriptProfileDescription (
   CMProfileRef prof,
   Str255 name,
   ScriptCode *code
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 166) to the profile whose profile name and script code are obtained.

name

A pointer to a name string. On return, the profile name.

code

A pointer to a script code. On return, the script code.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
The element data of the text description tag (which has the signature `'desc'` or constant `cmSigProfileDescriptionType`, defined in the `CMICCProfile.h` header file) specifies the profile name and script code. The `name` parameter returns the profile name as a Pascal string. Use this function so that your application does not need to obtain and parse the element data, which contains other information.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMApplication.h

CMLinkImage

Matches an image file with a device link profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMLinkImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 qual,
   CMProfileRef lnkProf,
   UInt32 lnkIntent
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

If a file with the same name already exists, it will be replaced if this parameter is set to true.

qual

The optional quality for the match—normal, draft or best (`cmNormalMode`, `cmDraftMode`, or `cmBestMode`).

lnkProf

The device link profile for the match.

lnkIntent

The rendering intent for the match—perceptual intent, relative colorimetric intent, saturation i ntent , or absolute colorimetric intent (`cmPerceptual`, `cmRelativecolorimetric`, `cmSaturation`, or `cmAbsoluteColorimetric`).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMMatchImage

Color matches an image file. (Deprecated in Mac OS X v10.5.)

```
CMError CMMatchImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 qual,
   CMProfileRef srcProf,
   UInt32 srcIntent,
   CMProfileRef dstProf
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

qual

The optional quality for the match—normal, draft or best (`cmNormalMode`, `cmDraftMode`, or `cmBestMode`).

srcProf

The optional source profile for the match.

srcIntent

The rendering intent for the match—perceptual intent, relative colorimetric intent, saturation i ntent , or absolute colorimetric intent (`cmPerceptual`, `cmRelativecolorimetric`, `cmSaturation`, or `cmAbsoluteColorimetric`).

dstProf

The destination profile for the match.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMNewProfileSearch

Searches the ColorSync Profiles folder and returns a list of 2.x profiles that match the search specification. (Deprecated in Mac OS X v10.5.)

```
CMError CMNewProfileSearch (
   CMSearchRecord *searchSpec,
   void *refCon,
   UInt32 *count,
   CMProfileSearchRef *searchResult
);
```

Parameters

searchSpec

A pointer to a search specification. For a description of the information you can provide in a search record of type `CMSearchRecord` to define the search, see `CMSearchRecord` (page 173). See the QuickDraw Reference for a description of the PixMap data type.

refCon

An untyped pointer to arbitrary data supplied by your application. `CMNewProfileSearch` passes this data to your filter routine. For a description of the filter routine, see the function `CMProfileFilterProcPtr` (page 105).

count

A pointer to a profile count. On return, a one-based count of profiles matching the search specification.

searchResult

A pointer to a search result reference. On return, a reference to the profile search result list. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 168). See the QuickDraw Reference for a description of the PixMap data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMNewProfileSearch` function sets up and defines a new search identifying through the search record the elements that a profile must contain to qualify for inclusion in the search result list. The function searches the ColorSync profiles folder for version 2.x profiles that meet the criteria and returns a list of these profiles in an internal private data structure whose reference is returned to you in the `searchResult` parameter.

You must provide a search record of type `CMSearchRecord` identifying the search criteria. You specify which fields of the search record to use for any given search through a search bit mask whose value you set in the search record's `searchMask` field.

Among the information you can provide in the search record is a pointer to a filter function to use to eliminate profiles from the search based on additional criteria not defined by the search record. The search result reference is passed to the filter function after the search is performed. For a description of the filter function and its prototype, see the function `CMProfileFilterProcPtr` (page 105).

Your application cannot directly access the search result list. Instead, you pass the returned search result list reference to other search-related functions that allow you to use the result list.

When your application has completed its search, it should call the function `CMDisposeProfileSearch` (page 285) to free the private memory allocated for the search.

To obtain a reference to a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfile` (page 302). To obtain the file specification for a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfileFileSpec` (page 302). To update the search result list, use the function `CMUpdateProfileSearch` (page 308). To free the private memory allocated for a profile search after your application has completed the search, use the function `CMDisposeProfileSearch` (page 285).

Version Notes
The `CMNewProfileSearch` function does not take full advantage of the optimized profile searching available starting with ColorSync version 2.5. Use `CMIterateColorSyncFolder` (page 57) instead.

This function is not recommended for use in ColorSync 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMProfileIdentifierFolderSearch

Searches the ColorSync Profiles folder and returns a list of profile references, one for each profile that matches the specified profile identifier. (Deprecated in Mac OS X v10.5.)

```
CMError CMProfileIdentifierFolderSearch (
   CMProfileIdentifierPtr ident,
   UInt32 *matchedCount,
   CMProfileSearchRef *searchResult
);
```

Parameters

ident
A pointer to a profile identifier structure specifying the profile to search for.

matchedCount
A pointer to a value of type `unsigned long`. On return, the one-based count of profiles that match the specified profile identifier. The count is typically 0 or 1, but can be higher.

searchResult
A pointer to a search result reference of type `CMProfileSearchRef` (page 168). On return, a reference to the profile search result list. Only version 2.x profiles are included in the profile search result.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261). It is not an error condition if this function finds no matching profiles. It returns an error only if a File Manager or other low-level system error occurs.

Discussion
When your application or device driver processes an image, it can keep a list of profile references for each profile it encounters in the image. Each time it encounters an embedded profile identifier, your application can call the function `CMProfileIdentifierListSearch` (page 299) to see if there is already a matching profile reference in its list. If not, it can call the `CMProfileIdentifierFolderSearch` function to see if the profile is located in the ColorSync Profiles folder.

Although there should typically be at most one profile in the ColorSync Profiles folder that matches the profile identifier, two or more profiles with different filenames may qualify.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

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

Declared In
`CMApplication.h`

CMProfileIdentifierListSearch

Searches a list of profile references and returns a list of all references that match a specified profile identifier. (Deprecated in Mac OS X v10.5.)

```
CMError CMProfileIdentifierListSearch (
   CMProfileIdentifierPtr ident,
   CMProfileRef *profileList,
   UInt32 listSize,
   UInt32 *matchedCount,
   CMProfileRef *matchedList
);
```

Parameters

ident

A pointer to a profile identifier. The function looks for profile references in `profileList` that match the profile described by this identifier. For information on how a profile identifier match is determined, see `CMProfileIdentifier` (page 162).

profileList

A pointer to a list of profile references to search.

listSize

The number of profile references in `profileList`.

matchedCount

A pointer to a count of matching profile references. If you set `matchedList` to `NULL`, On return `matchedCount` specifies the number of references in `profileList` that match `ident`. The count is typically 0 or 1, but can be higher.

If you do not set `matchedList` to `NULL`, on input you set `matchedCount` to the maximum number of matching references to be returned in `matchedList`. On return, the value of `matchedCount` specifies the actual number of matching references returned, which is always equal to or less than the number passed in.

matchedList

A pointer to a list of profile references. If you set `matchedList` to `NULL` on input, On return nothing is returned in the parameter, and the actual number of matching references is returned in `matchedCount`.

If you do not set `matchedList` to `NULL` on input, it is treated as a pointer to allocated memory. On return, the allocated memory will contain a list, in no particular order, of profile references that match `ident`. Only version 2.x profiles are included in the profile search result. The number of references in the list is equal to or less than the value you pass in the `matchedCount` parameter. You must allocate enough memory for `matchedList` to store the requested number of profile references.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261). It is not an error condition if the `CMProfileIdentifierListSearch` function finds no matching profiles. The function returns an error only if a Memory Manager or other low-level system error occurs.

Discussion
When your application or device driver processes an image, it can keep a list of profile references for each unique profile or profile identifier it encounters in the image. Each time it encounters an embedded profile identifier, your application can call the `CMProfileIdentifierListSearch` function to see if there is already a matching profile reference in the list. Although your list of profile references would normally contain at most one reference that matches the profile identifier, it is possible to have two or more matches. For information on how a profile identifier match is determined, see `CMProfileIdentifier` (page 162).

If no matching profile is found in the list, your application can call the function `CMProfileIdentifierFolderSearch` (page 298) to see if a matching profile can be found in the ColorSync Profiles folder.

To determine the amount of memory needed for the list of profile references that match a profile identifier, your application may want to call `CMProfileIdentifierListSearch` twice. The first time, on input you set `matchedList` to `NULL` and ignore `matchedCount`. On return, `matchedCount` specifies the number of matching profiles. You then allocate enough memory to hold that many profile references (or a smaller number if you do not want all the references) and call `CMProfileIdentifierListSearch` again. This time you set `matchedList` to a pointer to the allocated memory and set `matchedCount` to the number of references you wish to obtain. To allocate memory, you use code such as the following:

```
myProfileRefListPtr = NewPtr(sizeof(CMProfileRef) * matchedCount);
```

If your application is interested in obtaining only the first profile that matches the specified profile, you need call `CMProfileIdentifierListSearch` only once. To do so, you just allocate enough memory to store one profile reference, set `matchedList` to point to that memory (or just set `matchedList` to point to a local variable), and set `matchedCount` to 1. On return, if `matchedCount` still has the value 1, then `CMProfileIdentifierListSearch` found a matching profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMProofImage

Proofs an image. (Deprecated in Mac OS X v10.5.)

```
CMError CMProofImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 qual,
   CMProfileRef srcProf,
   UInt32 srcIntent,
   CMProfileRef dstProf,
   CMProfileRef prfProf
);
```

Parameters

specFrom

The destination profile for the match. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

qual

The optional quality for the match—normal, draft or best (`cmNormalMode`, `cmDraftMode`, or `cmBestMode`).

srcProf

The optional source profile for the match.

srcIntent

The rendering intent for the match—perceptual intent, relative colorimetric intent, saturation i ntent , or absolute colorimetric intent (`cmPerceptual`, `cmRelativecolorimetric`, `cmSaturation`, or `cmAbsoluteColorimetric`).

dstProf

The destination profile for the match.

prfProf

The proof profile for the match between the destination and proof profiles.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMSearchGetIndProfile

Opens the profile corresponding to a specific index into a specific search result list and obtains a reference to that profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMSearchGetIndProfile (
   CMProfileSearchRef search,
   UInt32 index,
   CMProfileRef *prof
);
```

Parameters

search

A reference to the profile search result list containing the profile whose reference you want to obtain. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 168). See the QuickDraw Reference for a description of the PixMap data type.

index

The position of the profile in the search result list. This value is specified as a one-based index into the set of profiles of the search result. The index must be less than or equal to the value returned as the `count` parameter of the `CMNewProfileSearch` function or the `CMUpdateProfileSearch` function; otherwise `CMSearchGetIndProfile` returns a result code of `cmIndexRangeErr`.

prof

A pointer to a profile reference of type `CMProfileRef` (page 166). On return, the reference refers to the profile associated with the specified index. See the QuickDraw Reference for a description of the PixMap data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Before your application can call the `CMSearchGetIndProfile` function, it must call the function `CMNewProfileSearch` (page 296) to perform a profile search and produce a search result list. The search result list is a private data structure maintained by the ColorSync Manager. After your application has finished using the profile reference, it must close the reference by calling the function `CMCloseProfile` (page 26).

Version Notes

This function is not recommended for use in ColorSync 2.5.

Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 57) for profile searching.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMSearchGetIndProfileFileSpec

Obtains the file specification for the profile at a specific index into a search result. (Deprecated in Mac OS X v10.5.)

```
CMError CMSearchGetIndProfileFileSpec (
   CMProfileSearchRef search,
   UInt32 index,
   FSSpec *spec
);
```

Parameters

search

A reference to the profile search result containing the profile whose file specification you want to obtain. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 168). See the QuickDraw Reference for a description of the PixMap data type.

index

The index of the profile whose file specification you want to obtain. This is a one-based index into a set of profiles in the search result list. The index must be less than or equal to the value returned as the `count` parameter of the `CMNewProfileSearch` function or the `CMUpdateProfileSearch` function; otherwise `CMSearchGetIndProfile` returns a result code of `cmIndexRangeErr`.

profileFile

A pointer to a file specification. On return, this parameter points to a file specification for the profile at the location specified by `index`. See the QuickDraw Reference for a description of the PixMap data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

Before your application can call the `CMSearchGetIndProfileFileSpec` function, it must call the function `CMNewProfileSearch` (page 296) to perform a profile search and produce a search result list. The search result list is a private data structure maintained by ColorSync.

The `CMSearchGetIndProfileFileSpec` function obtains the Macintosh file system file specification for a profile at a specific index in the search result list.

Version Notes

This function is not recommended for use in ColorSync 2.5.

Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 57) for profile searching.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMSetDefaultProfileBySpace

Sets the default profile for the specified color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetDefaultProfileBySpace (
   OSType dataColorSpace,
   CMProfileRef prof
);
```

Parameters

dataColorSpace

A four-character identifier of type `OSType`. You pass a color space signature that identifies the color space you wish to set the default profile for. The currently-supported values are `cmRGBData`, `cmCMYKData`, `cmLabData`, and `cmXYZData`. These constants are a subset of the constants described in "Color Space Signatures" (page 210). If you supply a value that is not supported, the `CMGetDefaultProfileBySpace` function returns an error value of `paramErr`.

prof

A profile reference. Before calling `CMSetDefaultProfileBySpace`, set the reference to specify the default profile for the color space. The profile must be file-based; otherwise, the function returns a `CMInvalidProfileLocation` error.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

The `CMSetDefaultProfileBySpace` function currently supports the RGB, CMYK, Lab, and XYZ color spaces. The signature constants for these color spaces (shown above with the `dataColorSpace` parameter description) are described in "Color Space Signatures" (page 210). Support for additional color spaces may be provided in the future. `CMSetDefaultProfileBySpace` returns a value of `paramErr` if you pass a color space constant it does not currently support.

Note that a user can also use the ColorSync control panel to specify a default profile for the RGB and CMYK color spaces.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMSetDefaultProfileByUse

Sets values for device profile settings. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetDefaultProfileByUse (
   OSType use,
   CMProfileRef prof
);
```

Parameters

use

A value that specifies the device type for which to set the profile.

prof

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMSetDeviceProfiles

Changes the profiles used by a given device. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetDeviceProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   const CMDeviceProfileScope *profileScope,
   const CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass
The device class for the device whose profiles you want to set. See "Device Classes" (page 220) for a list of the constants you can supply.

deviceID
The device ID for the device whose profiles you want to set.

profileScope
A pointer to the structure defining the scope these profiles pertain to.

deviceProfiles
A pointer to the profile array that contains replacements for the factory profiles. You don't have to replace all the original profiles with this call. The array can contain as few as one profile or as many profiles as there are in the original factory array. You supply only those profiles you want to replace. Profiles are replaced by ID.

Return Value
A `CMError` value. If you pass a n invalid `CMDeviceClass` or `CMDeviceID`, the function returns `CMInvalidDeviceClass` or `CMInvalidDeviceID`. See "ColorSync Manager Result Codes" (page 261).

Discussion
This function provides a way to change the profiles used by a given device. It can be called after device registration by calibration applications to reset a device's profiles from factory defaults to calibrated profiles. In order for this call to be made successfully, the caller must pass the `CMDeviceClass` and `CMDeviceID` device being calibrated. (You can call the function `CMIterateColorDevices` to find available device classes and IDs.).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

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

Declared In
`CMDeviceIntegration.h`

CMSetIndImageProfile

Sets a specific embeded profile for a given image. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetIndImageProfile (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 index,
   CMProfileRef prof
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

index

The numeric index of the profile to set.

prof

The profile to set at the index designated by the `index` parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMSetSystemProfile

Sets the current system profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetSystemProfile (
   const FSSpec *profileFileSpec
);
```

Parameters

profileFileSpec

A pointer to a file specification structure. Before calling `CMSetSystemProfile`, set the structure to specify the desired system profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

By default, a standard RGB profile is configured as the system profile. By calling the `CMSetSystemProfile` function, your application can specify a new system profile. You can configure only a display device profile as the system profile.

Version Notes

Starting with version 2.5, use of the system profile has changed.

The function `CMSetSystemProfile` does not retrieve video card gamma data (introduced in ColorSync version 2.5) to set the video card; use the function `CMSetProfileByAVID` (page 72) instead.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMUnembedImage

Removes any ICC profiles embeded in an image. (Deprecated in Mac OS X v10.5.)

```
CMError CMUnembedImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If his parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMUpdateProfileSearch

Searches the ColorSync Profiles folder and updates an existing search result obtained originally from the `CMNewProfileSearch` function. (Deprecated in Mac OS X v10.5.)

```
CMError CMUpdateProfileSearch (
   CMProfileSearchRef search,
   void *refCon,
   UInt32 *count
);
```

Parameters

search

A reference to a search result list returned to your application when you called the `CMNewProfileSearch` function. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 168). See the QuickDraw Reference for a description of the PixMap data type.

refCon

A pointer to a reference constant for application data passed as a parameter to calls to the filter function specified by the original search specification. For a description of the filter function, see the function `CMProfileFilterProcPtr` (page 105).

count

A pointer to a profile count. On return, if the function result is `noErr`, a one-based count of the number of profiles matching the original search specification passed to the `CMNewProfileSearch` function. Otherwise undefined.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
After a profile search has been set up and performed through a call to the `CMNewProfileSearch` function, the `CMUpdateProfileSearch` function updates the existing search result. You must use this function if the contents of the ColorSync Profiles folder have changed since the original search result was created.

The search update uses the original search specification, including the filter function indicated by the search record. Data given in the `CMUpdateProfileSearch` function's `refCon` parameter is passed to the filter function each time it is called.

Sharing a disk over a network makes it possible for modification of the contents of the ColorSync Profiles folder to occur at any time.

For a description of the function you call to begin a new search, see the function `CMNewProfileSearch` (page 296). That function specifies the filter function referred to in the description of the `refCon` parameter.

Version Notes
Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 57) for profile searching.

This function is not recommended for use in ColorSync 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMValidImage

Validates the specified image file. (Deprecated in Mac OS X v10.5.)

```
CMError CMValidImage (
   const FSSpec *spec
);
```

Parameters

spec

A file specification for the image file you want to validate. See the File Manager documentation for a description of the `FSSpec` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
This function validates the specified image file. ColorSync checks with any installed scripting plug-ins to see if they recognize the image's file format. If a scripting plug-in is found which recognizes the image's file format, `CMValidateImage` returns `noErr` . If the image's file format is not recognized, `CMValidateImage` returns the `cmInvalidImageFile` error.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CWNewLinkProfile

Creates a device link profile based on the specified set of profiles. (Deprecated in Mac OS X v10.5.)

```
CMError CWNewLinkProfile (
   CMProfileRef *prof,
   const CMProfileLocation *targetLocation,
   CMConcatProfileSet *profileSet
);
```

Parameters

prof

A pointer to an uninitialized profile reference of type `CMProfileRef` (page 166). On return, points to the new device link profile reference.

targetLocation

On return, a pointer to a location specification for the resulting profile. A device link profile cannot be a temporary profile: that is, it cannot have a location type of `cmNoProfileBase`.

profileSet

On input, a pointer to an array of profiles describing the processing to carry out. The array is in processing order—source through destination. For a description of the `CMConcatProfileSet` (page 128) data type, see `CMHeader` (page 139).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion

This discussion is accurate for versions of ColorSync prior to 2.5. See the version notes below for changes starting with version 2.5.

You can use this function to create a new single profile containing a set of profiles and pass the device link profile to the function `CWConcatColorWorld` (page 83) instead of specifying each profile in an array. A device link profile provides a means of storing in concatenated format a series of device profiles and non-device profiles that are used repeatedly in the same sequence.

The only way to use a device link profile is to pass it to the `CWConcatColorWorld` function as the sole profile specified by the array passed in the `profileSet` parameter.

The zero-based `keyIndex` field of the `CMConcatProfileSet` data structure specifies the index of the profile within the device link profile whose preferred CMM is used for the entire color-matching or color-checking session. The profile header's `CMMType` field specifies the preferred CMM for the specified profile. This CMM will fetch the profile elements necessary for the session.

The quality flag setting—indicating normal mode, draft mode, or best mode—specified by the first profile prevails for the entire session the quality flags of profiles that follow in the sequence are ignored. The quality flag setting is stored in the `flag` field of the profile header. See `CM2Header` (page 116) for more information on the use of flags.

The rendering intent specified by the first profile is used to color match to the second profile, the rendering intent specified by the second profile is used to color match to the third profile, and so on through the series of concatenated profiles.

The following rules govern the content and use of a device link profile:

- The first and last profiles you specify in the profiles array for a device link profile must be device profiles.
- You cannot specify a named color profile.

- You cannot include another device link profile in the series of profiles you specify in the profiles array.
- The only way to use a device link profile is to pass it to the `CWConcatColorWorld` function as the sole profile specified by the array passed in the `profileSet` parameter.
- You cannot embed a device link profile in an image.
- You cannot specify `NULL` to indicate the system profile.

This function privately maintains all the profile information required by the color world for color-matching and color-checking sessions. Therefore, after executing the `CWNewLinkProfile` function, you should call the `CMCloseProfile` (page 26) function for each profile used to build a device link profile (to dispose of each profile reference).

Version Notes
Note that starting with version 2.5, use of the system profile has changed.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

DisposeCMBitmapCallBackUPP

Disposes of a universal procedure pointer (UPP) to a bitmap callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMBitmapCallBackUPP (
   CMBitmapCallBackUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

DisposeCMConcatCallBackUPP

Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMConcatCallBackUPP (
   CMConcatCallBackUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMTypes.h`

DisposeCMFlattenUPP

Disposes of a universal procedure pointer (UPP) to a data-flattening callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMFlattenUPP (
   CMFlattenUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMTypes.h`

DisposeCMMIterateUPP

Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMMIterateUPP (
   CMMIterateUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
CMApplication.h

DisposeCMProfileAccessUPP

Disposes of a universal procedure pointer (UPP) to a profile-access callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMProfileAccessUPP (
   CMProfileAccessUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

DisposeCMProfileFilterUPP

Disposes of a universal procedure pointer (UPP) to a profile-filter callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMProfileFilterUPP (
   CMProfileFilterUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

DisposeCMProfileIterateUPP

Disposes of a universal procedure pointer (UPP) to a profile-iteration callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMProfileIterateUPP (
   CMProfileIterateUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMApplication.h`

InvokeCMBitmapCallBackUPP

Invokes a universal procedure pointer (UPP) to a bitmap callback. (Deprecated in Mac OS X v10.5.)

```
Boolean InvokeCMBitmapCallBackUPP (
   SInt32 progress,
   void *refCon,
   CMBitmapCallBackUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMBitmapCallBackProcPtr" (page 93) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

InvokeCMConcatCallBackUPP

Invokes a universal procedure pointer (UPP) to a progress-monitoring callback. (Deprecated in Mac OS X v10.5.)

```
Boolean InvokeCMConcatCallBackUPP (
   SInt32 progress,
   void *refCon,
   CMConcatCallBackUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMConcatCallBackProcPtr" (page 94) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

InvokeCMFlattenUPP

Invokes a universal procedure pointer (UPP) to a data-flattening callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMFlattenUPP (
   SInt32 command,
   long *size,
   void *data,
   void *refCon,
   CMFlattenUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMFlattenProcPtr" (page 96) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

InvokeCMMIterateUPP

Invokes a universal procedure pointer (UPP) to a a progress-monitoring callback for the `CMIterateCMMInfo` function. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMMIterateUPP (
   CMMInfo *iterateData,
   void *refCon,
   CMMIterateUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMMIterateProcPtr" (page 103) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
CMApplication.h

InvokeCMProfileAccessUPP

Invokes a universal procedure pointer (UPP) to a profile-access callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMProfileAccessUPP (
   SInt32 command,
   SInt32 offset,
   SInt32 *size,
   void *data,
   void *refCon,
   CMProfileAccessUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMProfileAccessProcPtr" (page 103) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

InvokeCMProfileFilterUPP

Invokes a universal procedure pointer (UPP) to a profile-filter callback. (Deprecated in Mac OS X v10.5.)

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMProfileFilterProcPtr" (page 105) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

InvokeCMProfileIterateUPP

Invokes a universal procedure pointer (UPP) to a profile-iteration callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMProfileIterateUPP (
   CMProfileIterateData *iterateData,
   void *refCon,
   CMProfileIterateUPP userUPP
);
```

Parameters

Return Value
A result code. See "ColorSync Manager Result Codes" (page 261).

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMProfileIterateProcPtr" (page 106) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMApplication.h`

NCMSetSystemProfile

Sets the location of a color profile. (Deprecated in Mac OS X v10.5.)

```
CMError NCMSetSystemProfile (
   const CMProfileLocation *profLoc
);
```

Parameters

profLoc

The location of the profile. Commonly a profile is disk-file based. However, the profile may be a file-based profile, a handle-based profile, or a pointer-based profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Discussion
Prior to ColorSync 2.6, the function for setting the system profile supported only the `FSSpec` file specification type as a way of specifying a profile. This function allows for greater flexibility when specifying a system profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

NCMUnflattenProfile

Unflattens a previouslyflattened profile. (Deprecated in Mac OS X v10.5.)

```
CMError NCMUnflattenProfile (
   CMProfileLocation *targetLocation,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

targetLocation

The location of the profile you want to unflatten. Commonly a profile is disk-file based. However, the profile may be a file-based profile, a handle-based profile, or a pointer-based profile.

proc

A user-defined procedure which is called during the unflatten operation.

refCon

A reference constant containing data specified by the calling application program.

preferredCMMnotfound

A return value indicating whether or not the CMM specified in the profile was found.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 261).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

NewCMBitmapCallBackUPP

Creates a new universal procedure pointer (UPP) to a bitmap callback. (Deprecated in Mac OS X v10.5.)

```
CMBitmapCallBackUPP NewCMBitmapCallBackUPP (
   CMBitmapCallBackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your bitmap callback function.

Return Value

The universal procedure pointer.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

NewCMConcatCallBackUPP

Creates a new universal procedure pointer (UPP) to a progress-monitoring callback. (Deprecated in Mac OS X v10.5.)

```
CMConcatCallBackUPP NewCMConcatCallBackUPP (
   CMConcatCallBackProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your progress-monitoring callback function.

Return Value
The universal procedure pointer.

Discussion
The callback protects against the appearance of a stalled machine during lengthy color world processing. If a CMM takes more than several seconds to process the information and create a color world, it will call the callback, if one is provided, and pass it the `refCon` provided. Passed to the functions `NCWNewLinkProfile` or `NCWConcatColorWorld` function .

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

NewCMFlattenUPP

Creates a new universal procedure pointer (UPP) to a data-flattening callback. (Deprecated in Mac OS X v10.5.)

```
CMFlattenUPP NewCMFlattenUPP (
   CMFlattenProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your data-flattening callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
CMTypes.h

NewCMMIterateUPP

Creates a new universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function. (Deprecated in Mac OS X v10.5.)

```
CMMIterateUPP NewCMMIterateUPP (
   CMMIterateProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your progress-monitoring callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
CMApplication.h

NewCMProfileAccessUPP

Creates a new universal procedure pointer (UPP) to a profile-access callback. (Deprecated in Mac OS X v10.5.)

```
CMProfileAccessUPP NewCMProfileAccessUPP (
   CMProfileAccessProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your profile-access callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

NewCMProfileFilterUPP

Creates a new universal procedure pointer (UPP) to a profile-filter callback. (Deprecated in Mac OS X v10.5.)

```
CMProfileFilterUPP NewCMProfileFilterUPP (
   CMProfileFilterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your profile-filter callback function.

Return Value

The universal procedure pointer.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMTypes.h`

NewCMProfileIterateUPP

Creates a new universal procedure pointer (UPP) to a profile-iteration callback. (Deprecated in Mac OS X v10.5.)

```
CMProfileIterateUPP NewCMProfileIterateUPP (
   CMProfileIterateProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your profile-iteration callback function.

Return Value

The universal procedure pointer.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMApplication.h`

Unsupported Functions

This section lists functions that are unsupported in Mac OS X. Table B-1 provides information on what you should do in place of using these functions.

Table B-1 Porting notes for unsupported ColorSync Manager functions

Unsupported functions	Porting notes
BeginMatching	BeginMatching is defined only if OLDROUTINENAMES is defined during compile time. Additionally, it uses 1.0 profiles, which are no longer supported. Use NCMBeginMatching (along with 2.0 profiles) instead.
CMAccelerationCalculateData	This function was used only by CMMs wishing to support hardware acceleration. With the advent of PowerPC chips, it no longer provides performance benefits over software implementations.
CMAccelerationLoadTables	This function was used only by CMMs wishing to support hardware acceleration. With the advent of PowerPC chips, it no longer provides performance benefits over software implementations.
CMBeginMatching	CMBeginMatching uses 1.0 profiles to establish onscreen matching. These profiles will not be supported. Use NCMBeginMatching instead.
CMCheckBitmap	Use CWCheckBitmap instead. CMCheckBitmap is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMCheckColors	Use CWCheckColors instead. CMCheckColors is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMCheckPixMap	CMCheckPixMap is an API to CMMs. Use CWCheckPixMap instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMConcatenateProfiles	CMConcatenateProfiles is an API for CMMs, and it uses 1.0 profiles.
CMConcatInit	CMConcatInit is an API to CMMs. Application developers should avoid calling CMMs directly, since they may or may not support a given API. To establish a color world using a set of profiles, use CWConcatColorWorld.
CMConvertProfile2to1	ColorSync 1.0 profiles will no longer be supported, so CMConvertProfile2to1 has no utility.
CMDeleteDeviceProfile	CMDeleteDeviceProfile is an API to the 1.0 Profile Responder component, which is no longer supported.

Unsupported functions	Porting notes
CMDrawMatchedPicture	CMDrawMatchedPicture uses 1.0 profiles to match the PICT data, and these profiles will no longer be supported. Use NCMDrawMatchedPicture, which uses 2.0 profiles.
CMFixedXYZToXYZ	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertFixedXYZToXYZ.
CMGetIndexedProfile	CMGetIndexedProfile is an API to the 1.0 Profile Responder component, which is no longer supported. Services for searching and indexing 2.0 profiles are supported via CMNewProfileSearch, CMSearchGetIndProfile, and other search routines.
CMGetProfile	CMGetProfile is an API (for 1.0 profiles) to the Profile Responder component, which is no longer supported. Access to 2.0 profiles is supported via CMGetSystemProfile, CMOpenProfile, and the search routines.
CMGetProfileAdditionalDataOffset	CMGetProfileAdditionalDataOffset provides access to data within 1.0 profiles, and these profiles will no longer be supported.
CMGetProfileName	CMGetProfileName gets the name of 1.0 profiles, which will no longer be supported.
CMHLSToRGB	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertHLSToRGB.
CMHSVToRGB	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertHSVToRGB.
CMInit	This was the initialization routine for old style color worlds. Use NCMInit to create color worlds.
CMLabToXYZ	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertLabToXYZ.
CMLuvToXYZ	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertLuvToXYZ.
CMMatchBitmap	CMMatchBitmap is an API to CMMs. Use CWMatchBitMap instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMatchColors	CMMatchColors is an API to CMMs. Use CWMatchColors instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMatchPixMap	CMMatchPixMap is an API to CMMs. Use CWMatchPixMap instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.

Unsupported functions	Porting notes
CMMCheckBitmap	Use CWCheckBitMap instead. CMMCheckBitmap is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMCheckColors	Use CWCheckColors instead. CMMCheckColors is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMCheckPixMap	Use CWCheckPixMap instead. CMMCheckPixMap is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMClose	This is a Component Manager wrapper function which only applies to ColorSync on Mac OS 8 and 9.
CMMConcatenateProfiles	This is an API to CMMs, which aren\qt supported in Carbon.
CMMConcatInit	This is an API to CMMs, which aren\qt supported in Carbon. To establish a color world using a set of profiles, use CWConcatColorWorld.
CMMFlattenProfile	CMMFlattenProfile is an API to CMMs. Use CMFlattenProfile instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetCMMInfo	This is a Component Manager wrapper function which only applies to ColorSync on Mac OS 8 and 9.
CMMGetIndNamedColorValue	CMMGetIndNamedColorValue is an API to CMMs. Use CMGetIndNamedColorValue instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetNamedColorIndex	CMMGetNamedColorIndex is an API to CMMs. Use CMGetNamedColorIndex instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetNamedColorInfo	CMMGetNamedColorInfo is an API to CMMs. Use CMGetNamedColorInfo instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetNamedColorName	CMMGetNamedColorName is an API to CMMs. Use CMGetNamedColorName instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetNamedColorValue	CMMGetNamedColorValue is an API to CMMs. Use CMGetNamedColorValue instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.

Unsupported functions	Porting notes
CMMGetPS2ColorRendering	CMMGetPS2ColorRendering is an API to CMMs; use CMGetPS2ColorRendering instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetPS2ColorRenderingIntent	CMMGetPS2ColorRenderingIntent is an API to CMMs. Use CMGetPS2ColorRenderingIntent instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetPS2ColorRenderingVMSize	CMMGetPS2ColorRenderingVMSize is an API to CMMs; use CMGetPS2ColorRenderingVMSize instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMGetPS2ColorSpace	CMMGetPS2ColorSpace is an API to CMMs. Use CMGetPS2ColorSpace instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMInit	This was the initialization routine for old style color worlds. Use NCMInit to create color worlds.
CMMMatchBitmap	Use CWMatchBitMap instead. CMMMatchBitmap is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMMatchColors	Use CWMatchColors instead. CMMMatchColors is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMMatchPixMap	Use CWMatchPixMap instead. CMMMatchPixMap is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMNewLinkProfile	Use CWNewLinkProfile instead. CMMNewLinkProfile is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.
CMMOpen	This is a Component Manager wrapper function which only applies to ColorSync on Mac OS 8 and 9.
CMMUnflattenProfile	CMMUnflattenProfile is an API to CMMs. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMMValidateProfile	CMMValidateProfile is an API to CMMs. Use CMValidateProfile instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.
CMNewLinkProfile	CMNewLinkProfile is an API to CMMs. Use CWNewLinkProfile instead. Application developers should avoid calling CMMs directly, since they may or may not support a given API.

Unsupported functions	Porting notes
CMRGBToGray	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertRGBToGray.
CMRGBToHLS	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertRGBToHLS.
CMRGBToHSV	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertRGBToHSV.
CMSetProfile	CMSetProfile is an API (for 1.0 profiles) to the Profile Responder component, which is no longer supported. Access to 2.0 profiles is supported via CMSetSystemProfile, CMNewProfile, and other routines.
CMSetProfileDescription	CMSetProfileDescription is an API to the Profile Responder component, which is no longer supported. It also operated on 1.0 profiles, which are no longer supported. Access to internal profile data for 2.0 profiles is supported via CMSetProfileElement.
CMUnflattenProfile	Because this function unflattens only those profiles based on FSSpec structures, you should use NCMUnflattenProfile instead.
CMUseProfileComment	CMUseProfileComment embeds 1.0 profiles in the PICT data. These profiles will no longer be supported. Use NCMUseProfileComment instead.
CMXYZToFixedXYZ	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertXYZToFixedXYZ.
CMXYZToLab	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertXYZToLab.
CMXYZToLuv	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertXYZToLuv.
CMXYZToYxy	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertXYZToYxy.
CMYxyToXYZ	This function is simply glue to the old CMConversion Component. The preferred access method to this function is CMConvertYxyToXYZ.
ConcatenateProfiles	ConcatenateProfiles operates on 1.0 profiles, which are no longer supported. Concatenation is supported for 2.0 profiles via CWNewLinkProfile and other APIs.
CWNewColorWorld	CWNewColorWorld takes as parameters 1.0 profiles, which will no longer be supported. Use NCWNewColorWorld instead.
DeleteDeviceProfile	DeleteDeviceProfile deletes 1.0 profiles, which will no longer be supported.

Unsupported functions	Porting notes
DisposeOldCalibrateUPP	This function was intended for use only by the ColorSync Manager itself and not by applications. Applications should have no need to use this function.
DisposeOldCanCalibrateUPP	This function was intended for use only by the ColorSync Manager itself and not by applications. Applications should have no need to use this function.
DrawMatchedPicture	DrawMatchedPicture uses 1.0 profiles, which are obsolete. Use NCMDrawMatchedPicture (which supports 2.0 profiles) instead.
EnableMatching	EnableMatching is a valid API only if OLDROUTINENAMES is defined for a given compile. Use CMEnableMatchingComment for full compatibility.
EndMatching	EndMatching is defined if OLDROUTINENAMES is used during compilation. Use CMEndMatching for full compatibility.
GetIndexedProfile	GetIndexedProfile provides access to 1.0 profiles, which will no longer be supported. CMNewProfileSearch and CMSearchGetIndProfile provide enhanced access to 2.0 profiles.
GetProfile	GetProfile provides access to 1.0 profiles, which will no longer be supported.
GetProfileAdditionalDataOffset	GetProfileAdditionalDataOffset is a data accessor for 1.0 profiles, which will no longer be supported.
GetProfileName	GetProfileName is an accessor for 1.0 profiles, which will no longer be supported. Access to 2.0 profile data is supported by CMGetScriptProfileDescription and CMGetProfileElement.
InvokeOldCalibrateUPP	This function was intended for use only by the ColorSync Manager itself and not by applications. Applications should have no need to use this function.
InvokeOldCanCalibrateUPP	This function was intended for use only by the ColorSync Manager itself and not by applications. Applications should have no need to use this function.
NCMInit	This is a Component Manager wrapper function which only applies to ColorSync on Mac OS 8 and 9.
NCMMConcatInit	This is an API to CMMs, which aren't supported in Carbon. To establish a color world using a set of profiles, use CWConcatColorWorld.
NCMMInit	This was the initialization routine for old style color worlds. Use NCMInit to create color worlds.
NCMMNewLinkProfile	Use CWNewLinkProfile instead. NCMMNewLinkProfile is an API to CMMs -- Component Manager structures that Mac OS X does not support. In Carbon, CMMs are replced by opaque structures of type CFBundle.

Unsupported functions	Porting notes
NewOldCalibrateUPP	This function was intended for use only by the ColorSync Manager itself and not by applications. Applications should have no need to use this function.
NewOldCanCalibrateUPP	This function was intended for use only by the ColorSync Manager itself and not by applications. Applications should have no need to use this function.
SetProfile	SetProfile uses 1.0 profiles, which will no longer be supported. To set the System profile, use NCMSetSystemProfile.
SetProfileDescription	SetProfileDescription provides access to 1.0 profiles, which will no longer be supported. To set the description of a 2.0 profile, use CMSetProfileElement.
UseProfile	UseProfile allows 1.0 profiles to be used within PICT data streams. These profiles will no longer be supported.

Document Revision History

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

Date	Notes
2005-06-04	Corrected a typo in the parameter list for CWCheckColors.
2004-04-22	Added documentation for the following functions new to Mac OS X v 10.3: `CMCopyProfileDescriptionString` (page 29), `CWFillLookupTexture` (page 85), and `CMMakeProfile` (page 60).
	Moved device integration errors from Constants section to "ColorSync Manager Result Codes" (page 261) and added documentation for them.
	Reorganized the functions and renamed several of the function groups to better reflect the usage of the functions.
	Removed data types and constants that no longer appear in the header files.
	Added abstracts for some data types and constant groups.
	Renamed many enumerations that were formerly named for the first constant in the group, to better reflect what the constants are used for.
2003-05-01	Added abstracts for many functions and callbacks.
2003-02-01	Updated formatting and linking. Added appendix listing unsupported functions.

Document Revision History

Index

D

E

F

G

I

K

M

N

O

P

Q

R

S

T

U

V