Managing Colors With ColorSync in Mac OS 9

(Legacy)

Mac OS 9 & Earlier > Unsupported

2003-02-01

Adobe, Acrobat, and PostScript are trademarks or registered trademarks of Adobe Systems Incorporated in the U.S. and/or other countries.

Contents

Figures, Tables, and Listings

About This Document

Important: The information in this document is obsolete and should not be used for new development.

This document describes ColorSync, the color management system from Apple Computer, Inc. that provides essential services for fast, consistent, and accurate color management. It also describes the ColorSync Manager, the application programming interface (API) to these services.

This Preface covers:

- What's in This Document (page 9)
- Conventions (page 10)
- Important Note on Code Listings (page 11)

For additional information about this document, see What's New.

What's in This Document

This document introduces ColorSync and the concepts of color management, shows how to use ColorSync in applications and device drivers, and provides an overview of developing color management modules (CMMs). It describes features available through ColorSync version 2.5. Most existing code written to use version 2.0 or 2.1 of the ColorSync Manager should continue to work with version 2.5 without modification.

Note: There are no changes to the ColorSync Manager API between version 2.5 and version 2.5.1, so this document is up-to-date for ColorSync 2.5.1.

This document includes the following sections, as well as a glossary and index.

- Overview of Color and Color Management Systems (page 13) provides a general introduction to color-management, defines terms such as profile, color space, and CMM, and serves as a primer for those unfamiliar with color management systems.
- Overview of ColorSync (page 27) provides an overview of ColorSync and the ColorSync Manager, including both user interface and API elements. It describes ColorSync's support for scripting, monitor calibration, the use of multiple processors, and other features.
- Developing ColorSync-Supportive Applications (page 49) describes how your application can use the ColorSync Manager to provide many color management services. It includes detailed code samples.
- Developing ColorSync-Supportive Device Drivers (page 109) describes how you can use the ColorSync Manager to create ColorSync-supportive drivers for peripherals such as input, output, and display devices.
- Developing Color Management Modules (page 121) describes how to create a color management module (CMM) component that ColorSync can use to match and check colors.

- Version and Compatibility Information (page 141) describes the `Gestalt` information, shared library version numbers, CMM version numbers, and ColorSync header files you use with different versions of the ColorSync Manager. It also describes CPU and system requirements. In addition, it describes backward compatibility between versions of the ColorSync Manager and the profile formats they use.
- What's New (page 151) lists the new features available with ColorSync 2.5 and provides links to new and revised material. It includes a summary of new and changed code listings, functions, data types, and constants. It also includes a list of features new to ColorSync version 2.1, as well as information on where to obtain documentation for other color-related technologies.

Conventions

This document uses the following conventions to help you locate information.

Version Notes

Functions and data types that are changed or not recommended in ColorSync version 2.5 generally contain a VERSION NOTES section that summarizes the changes or points to related information.

Special Fonts

All code listings, reserved words, and the names of actual data structures, constants, fields, parameters, and routines are shown in a monospaced font such as Letter Gothic (`this is Letter Gothic`).

Words that appear in boldface are key terms or concepts and are defined in the glossary.

Types of Notes

There are several types of notes used in this document.

Note: A note like this contains information that is interesting but possibly parenthetical to the main text.

Important: A note like this sets off information that is essential for an understanding of the main text.

Warning: Warnings like this indicate potential problems that you should be aware of as you design your application or device driver. Failure to heed these warnings could result in system crashes or loss of data.

Important Note on Code Listings

All code listings in this document are shown in C, except for listings that describe resources, which are shown in Rez-input format. Many listings are from the CSDemo application, which is available with the ColorSync 2.5 SDK. See Figures, Tables, and Listings for the locations of all code listings in this document.

Important: Although the listings in this document have been compiled and, to some degree, tested, Apple Computer does not promote the direct incorporation of these code samples into your application. For example, to make the code listings in this document more readable, only limited error handling is shown. You need to develop your own techniques for detecting and handling errors.

Overview of Color and Color Management Systems

This section provides a very brief description of ColorSync, the cross-platform color management system from Apple Computer, Inc. It then provides a general introduction to the basics of color and color management systems.

Read this section to learn about color perception, additive and subtractive color systems, how different peripheral devices represent color, and how color management systems maintain consistent color among devices. If you are already familiar with these concepts, you can skip ahead to "Overview of ColorSync" (page 13), which provides a detailed overview of ColorSync.

For more information on color theory and color spaces, see:

- Fred W. Billmeyer, Jr., and Max Saltzman. Principles of Color Technology, second edition. Wiley, 1981.
- James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. Computer Graphics: Principles and Practice, second edition. Addison-Wesley, 1990.
- Roy Hall. Illumination and Color in Computer Generated Imagery. New York: Springer-Verlag, 1988.
- R.W.G. Hunt. Measuring Colour, second edition. Prentice-Hall, 1991.
- Günther Wyszecki and W.S. Stiles. Color Science: Concepts and Methods, Quantitative Data and Formulae, second edition. A Wiley-Interscience Publication, 1982.

ColorSync

ColorSync is the platform-independent color management system from Apple Computer, Inc. ColorSync provides essential services for fast, consistent, and accurate desktop color calibration, proofing, and reproduction for the graphic arts, publishing, and printing industries. The ColorSync Manager is the application programming interface (API) to these services. ColorSync and the ColorSync Manager are described in detail in Overview of ColorSync (page 27). Color management systems are defined in Color Management Systems (page 25).

Color: A Brief Overview

Color is a sensation and, therefore, a subjective experience. The sensation of color is one component of the visual sensation, caused by the sensitivity of the human eye to light. Light can be perceived either directly from light sources (such as the sun, a fire, incandescent or fluorescent bulbs, television screens, and computer displays) or indirectly, when light from these sources is transmitted through or reflected by objects. Color sensation is also affected by how the brain processes information and is specific to each individual. Thus color perception is avery complex phenomenon.

The foundation of the color reproduction process is trichromatic color vision, which describes the capacity of the human eye to respond equally to two or more sets of stimuli having different visible spectra. This means that two or more visible spectra may exist that will be perceived as the same color, a phenomenon known as metamerism. Because of this property, spectral color reproduction, a very expensive and impractical process, can be replaced by trichromatic color reproduction, a process that is much cheaper and easier to control.

Trichromatic color reproduction induces the illusion of a color using various amounts of only three primary colors: either red, green, and blue mixed additively or cyan, magenta, and yellow mixed subtractively. Additive and subtractive colors are described in Additive and Subtractive Color (page 15). Trichromatic color reproduction is the fundamental mechanism used in the majority of color reproduction devices, from television, computer display and movie screens, to magazines, newspapers, large posters, and small pages printed on your desktop printer.

Computers enable us to control color digitally and many peripherals have been developed for acquiring, displaying, and reproducing color. As a result, there is a need for a mechanism to maintain color control in an environment that can include different computer operating systems and hardware, as well as a wide variety of devices and media connected to the computer.

In the Mac OS, the ColorSync Manager is the part of the operating system that provides color management. For a detailed description, see ColorSync Manager Overview (page 28).

Color Perception

The eye contains two types of receptors, cones and rods. The rods measure illumination and are not sensitive to color. The cones contain a chemical known as Rhodopsin, which is variously sensitive to reds and blues and has a default sensitivity to yellow. The color the eyes see in an object depends on how much red, green, and blue light is reflected to a small region in the back of the eye called the fovea, which contains a great majority of the cones present in the eye. Black is perceived when no light is reflected to the eye.

Even the conditions in which color is viewed greatly affect the perception of color. The light source and environment must be standardized for accurate viewing. When viewing colors, people in the graphic arts industry, for example, avoid fluorescent and tungsten lighting, use a particular illuminant that is similar to daylight, and proof against a neutral gray surface.

Color images frequently contain hundreds of distinctly different colors. To reproduce such images on a color peripheral device is impractical. However, a very broad range of colors can be visually matched by a mixture of three primary lights. This allows colors to be reproduced on a display by a mixture of red, green, and blue lights (the primary colors of the additive color space shown in Figure 2-4 (page 19)) or on a printer by a mixture of cyan, magenta, and yellow inks or pigments (the primary colors of the subtractive color space shown in Figure 2-4 (page 19)). Black is printed to increase contrast and make up for the deficiency of the inks (making black the key, or K, in CMYK).

Hue, Saturation, and Value (or Brightness)

Color is described as having three dimensions. These dimensions are hue, saturation, and value. Hue is the name of the color, which places the color in its correct position in the spectrum. For example, if a color is described as blue, it is distinguished from yellow, red, green, or other colors. Saturation refers to the degree of intensity in a color, or a color's strength. A neutral gray is considered to have zero saturation. A saturated red would have a color similar to apple red. Pink is an example of an unsaturated red. Value (or brightness) describes differences in the intensity of light reflected from or transmitted by a color image. The hue of an object may be blue, but the terms dark and light distinguish the value, or brightness, of one object from another. The 3-dimensional color spaces based on hue, saturation and value are described in HSV and HLS Color Spaces (page 17).

Additive and Subtractive Color

The additive color theory refers to the process of mixing red, green, and blue lights, which are each approximately one-third of the visible spectrum. Additive color theory explains how red, green, and blue light can be added to make white light. Red and green projected together produce yellow, red and blue produce magenta, and blue and green produce cyan. With red, blue, and green transmitted light, all the colors of the rainbow can be matched.

The subtractive color theory refers to the process of combining subtractive colorants such as inks or dyes. In this theory, various levels of cyan, magenta, and yellow absorb or subtract a portion of the spectrum of white light that is illuminating an object. The color of an object is the result of the color lights that are not absorbed by the object. An apple appears red because the surface of the apple absorbs the blue and green light.

Monitors use the additive color space, output printing devices use the subtractive color space.

Color Spaces

A color space describes an environment in which colors are represented, ordered, compared, or computed. A color space defines a one-, two-, three-, or four-dimensional environment whose components (or color components) represent intensity values. A color component is also referred to as a color channel. For example, RGB space is a three-dimensional color space whose stimuli are the red, green, and blue intensities that make up a given color; and red, green, and blue are color channels. Visually, these spaces are often represented by various solid shapes, such as cubes, cones, or polyhedra.

For additional information on color components, see Color-Component Values, Color Values, and Colors (page 23).

The ColorSync Manager directly supports several different color spaces to give you the convenience of working in whatever kind of color data most suits your needs. The ColorSync color spaces fall into several groups, or base families. They are:

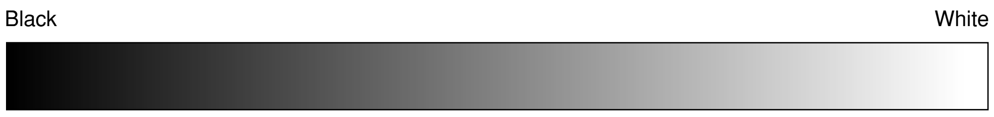
- gray spaces, used for grayscale display and printing; see Gray Spaces (page 16)
- RGB-based color spaces, used mainly for displays and scanners; see RGB-Based Color Spaces (page 16)
- CMYK-based color spaces, used mainly for color printing; see CMY-Based Color Spaces (page 18)
- device-independent color spaces, such as L*a*b, used mainly for color comparisons, color differences, and color conversion; see Device-Independent Color Spaces (page 19)
- named color spaces, used mainly for printing and graphic design; see Named Color Spaces (page 23)
- heterogeneous HiFi color spaces, also referred to as multichannel color spaces, primarily used in new printing processes involving the use of red-orange, green and blue, and also for spot coloring, such as gold and silver metallics; see Color-Component Values, Color Values, and Colors (page 23)

All color spaces within a base family are related to each other by very simple mathematical formulas or differ only in details of storage format.

Gray Spaces

Gray spaces typically have a single component, ranging from black to white, as shown in Figure 2-1 (page 16). Gray spaces are used for black-and-white and grayscale display and printing. A properly plotted gray space should have a fifty percent value as its midpoint.

Figure 2-1 Gray space



RGB-Based Color Spaces

The RGB space is a three-dimensional color space whose components are the red, green, and blue intensities that make up a given color. For example, scanners read the amounts of red, green, and blue light that are reflected from or transmitted through an image and then convert those amounts into digital values. Information displayed on a color monitor begins with digital values that are converted to analog signals for display on the monitor. The analog signals are transmitted to the phosphors on the face of the monitor, causing them to glow at various intensities of red, green, and blue (the combination of which makes up the required hue, saturation, and brightness of the desired colors).

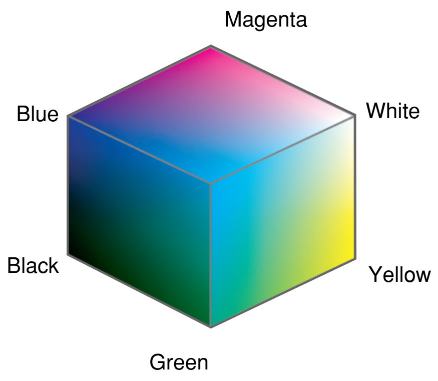
RGB-based color spaces are the most commonly used color spaces in computer graphics, primarily because they are directly supported by most color displays and scanners. RGB color spaces are device dependent and additive. The groups of color spaces within the RGB base family include

- RGB spaces
- HSV and HLS spaces

RGB Spaces

Any color expressed in RGB space is some mixture of three primary colors: red, green, and blue. Most RGB-based color spaces can be visualized as a cube, as in Figure 2-2 (page 16), with corners of black, the three primaries (red, green, and blue), the three secondaries (cyan, magenta, and yellow), and white.

Figure 2-2 RGB color space (Red corner is hidden from view)



sRGB Color Space

The sRGB color space is based on the ITU-R BT.709 standard. It specifies a gamma of 2.2 and a white point of 6500 degrees K. You can read more about sRGB space at the International Color Consortium site at <http://www.color.org/>. This space gives a complimentary solution to the current strategies of color management systems, by offering an alternate, device-independent color definition that is easier to handle for device manufacturers and the consumer market. sRGB color space can be used if no other RGB profile is specified or available. Starting with version 2.5, ColorSync provides full support for sRGB, including an sRGB profile.

Note that as an open architecture, ColorSync is not tied to the use of the sRGB color space and can support any RGB space that the user might prefer. For example, high end users with good quality reproduction devices may find that the sRGB space, which limits colors to the sRGB gamut, is too restrictive for their required color quality.

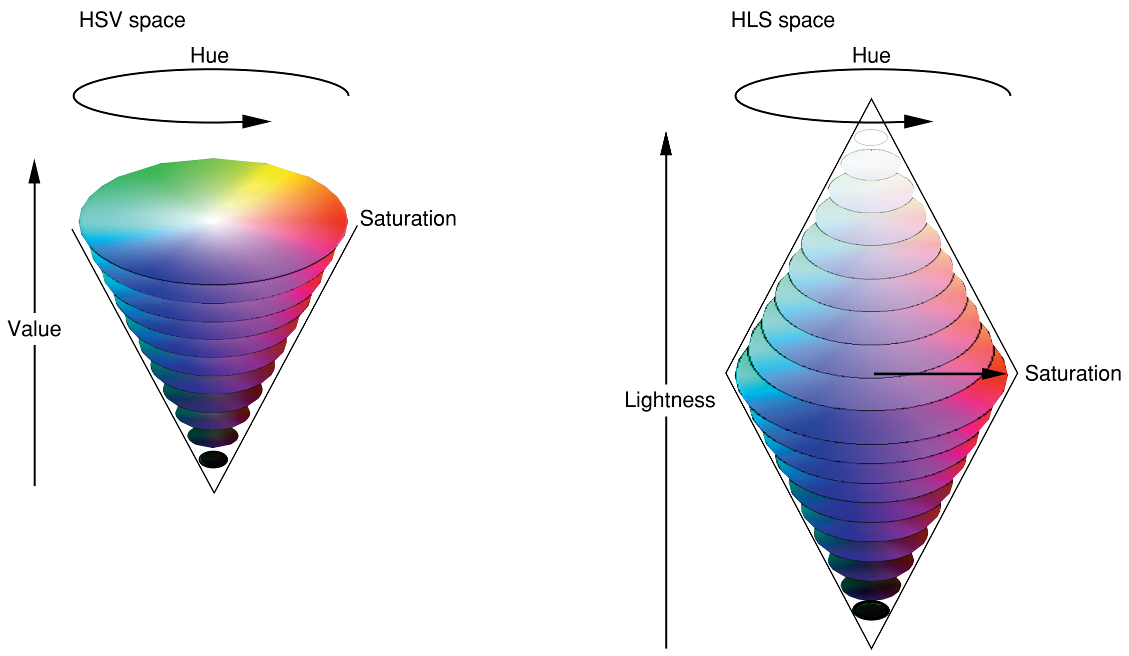
HSV and HLS Color Spaces

HSV space and HLS space are transformations of RGB space that can describe colors in terms more natural to an artist. The name HSV stands for hue, saturation, and value. (HSB space, or hue, saturation, and brightness, is synonymous with HSV space.) HLS stands for hue, lightness, and saturation. The two spaces can be thought of as being single and double cones, as shown in Figure 2-3 (page 18).

The components in HLS space are analogous, but not completely identical, to the components in HSV space:

- The hue component in both color spaces is an angular measurement, analogous to position around a color wheel. A hue value of 0 indicates the color red; the color green is at a value corresponding to 120, and the color blue is at a value corresponding to 240. Horizontal planes through the cones in Figure 2-3 (page 18) are hexagons; the primaries and secondaries (red, yellow, green, cyan, blue, and magenta) occur at the vertices of the hexagons.
- The saturation component in both color spaces describes color intensity. A saturation value of 0 (in the middle of a hexagon) means that the color is colorless (gray); a saturation value at the maximum (at the outer edge of a hexagon) means that the color is at maximum colorfulness for that hue angle and brightness.

Figure 2-3 HSV (or HSB) color space and HLS color space



- The value component in HSV describes the brightness. In both color spaces, a value of 0 represents the absence of light, or black. In HSV space, a maximum value means that the color is at its brightest. In HLS space, a maximum value for lightness means that the color is white, regardless of the current values of the hue and saturation components.

CMY-Based Color Spaces

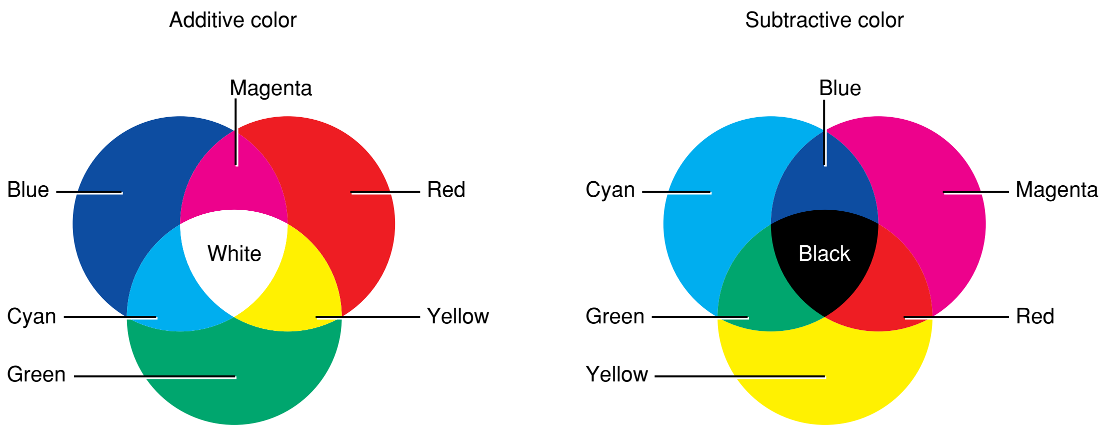
CMY-based color spaces are most commonly used in color printing systems. They are device dependent and subtractive in nature. The groups of color spaces within the CMY family include

- CMY, which is not very common except on low-end color printers
- CMYK, which models the way inks or dyes are applied to paper in printing

The name CMYK refers to cyan, magenta, yellow, and key (represented by black). Cyan, magenta, and yellow are the three primary colors in this color space, and red, green, and blue are the three secondaries. Theoretically black is not needed. However, when full-saturation cyan, magenta, and yellow inks are mixed equally on paper, the result is usually a dark brown, rather than black. Therefore, black ink is overprinted in darker areas to expand the dynamic range and give a better appearance. Printing with black ink makes it possible to use less cyan, magenta, and yellow ink. This may prevent saturation, especially on materials such as plain paper which cannot accept too much ink. Using black can also reduce the cost per page because cyan, magenta, and yellow inks are generally more expensive than black ink. It can also provide a sharper image, because a single dot of black ink is used in place of three dots of other inks.

Figure 2-4 (page 19) shows how additive and subtractive colors mix to form other colors.

Figure 2-4 Additive and subtractive colors



Theoretically, the relation between RGB values and CMY values in CMYK space is quite simple:

```
Cyan        = 1.0 - red
Magenta     = 1.0 - green
Yellow      = 1.0 - blue
```

(where red, green, and blue intensities are expressed as fractional values varying from 0 to 1). In reality, the process of deriving the cyan, magenta, yellow, and black values from a color expressed in RGB space is complex, involving device-specific, ink-specific, and even paper-specific calculations of the amount of black to add in dark areas (black generation) and the amount of other ink to remove (undercolor removal) where black is to be printed. Therefore, when ColorSync converts between CMYK and RGB color spaces, it uses an elaborate system of multi-dimensional lookup tables, which ColorSync knows how to interpret. This information is stored in profiles, which are defined in the section Color Conversion and Color Matching (page 23).

Device-Independent Color Spaces

Some color spaces can express color in a device-independent way. Whereas RGB colors vary with display and scanner characteristics, and CMYK colors vary with printer, ink, and paper characteristics, device-independent colors are not dependent on any particular device and are meant to be true representations of colors as perceived by the human eye. These color representations, called device-independent color spaces, result from work carried out by the Commission Internationale d'Eclairage (CIE) and for that reason are also called CIE-based color spaces.

The most common method of identifying color within a color space is a three-dimensional geometry. The three color attributes, hue, saturation, and brightness, are measured, assigned numeric values, and plotted within the color space.

Conversion from an RGB color space to a CMYK color space involves a number of variables. The type of printer or printing press, the paper stock, and the inks used all influence the balance between cyan, magenta, yellow, and black. In addition, different devices have different gamuts, or ranges of colors that they can produce. Because the colors produced by RGB and CMYK specifications are specific to a device, they're called device-dependent color spaces. Device color spaces enable the specification of color values that are directly related to their representation on a particular device.

Device-independent color spaces can be used as interchange color spaces to convert color data from the native color space of one device to the native color space of another device.

The CIE created a set of color spaces that specify color in terms of human perception. It then developed algorithms to derive three imaginary primary constituents of color—X, Y, and Z—that can be combined at different levels to produce all the color the human eye can perceive. The resulting color model, CIEXYZ, and other CIE color models form the basis for all color management systems. Although the RGB and CMYK values differ from device to device, human perception of color remains consistent across devices. Colors can be specified in the CIE-based color spaces in a way that is independent of the characteristics of any particular display or reproduction device. The goal of this standard is for a given CIE-based color specification to produce consistent results on different devices, up to the limitations of each device.

XYZ Space

There are several CIE-based color spaces, but all are derived from the fundamental XYZ space. The XYZ space allows colors to be expressed as a mixture of the three tristimulus values X, Y, and Z. The term tristimulus comes from the fact that color perception results from the retina of the eye responding to three types of stimuli. After experimentation, the CIE set up a hypothetical set of primaries, XYZ, that correspond to the way the eye's retina behaves.

The CIE defined the primaries so that all visible light maps into a positive mixture of X, Y, and Z, and so that Y correlates approximately to the apparent lightness of a color. Generally, the mixtures of X, Y, and Z components used to describe a color are expressed as percentages ranging from 0 percent up to, in some cases, just over 100 percent.

Other device-independent color spaces based on XYZ space are used primarily to relate some particular aspect of color or some perceptual color difference to XYZ values.

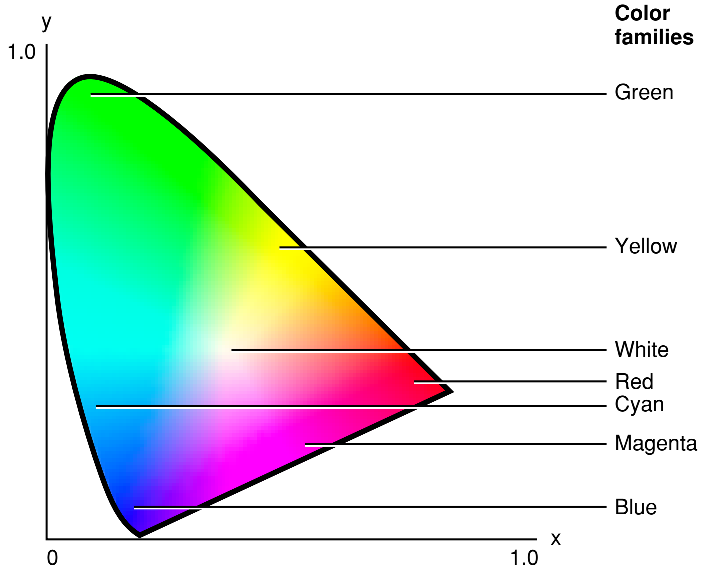
Yxy Space

Yxy space expresses the XYZ values in terms of x and y chromaticity coordinates, somewhat analogous to the hue and saturation coordinates of HSV space. The coordinates are shown in the following formulas, used to convert XYZ into Yxy:

```
Y = Y
x = X / (X+Y+Z)
y = Y / (X+Y+Z)
```

Note that the Z tristimulus value is incorporated into the new coordinates and does not appear by itself. Since Y still correlates to the lightness of a color, the other aspects of the color are found in the chromaticity coordinates x and y. This allows color variation in Yxy space to be plotted on a two-dimensional diagram. Figure 2-5 (page 21) shows the layout of colors in the x and y plane of Yxy space.

Figure 2-5 Yxy chromaticities in the CIE color space

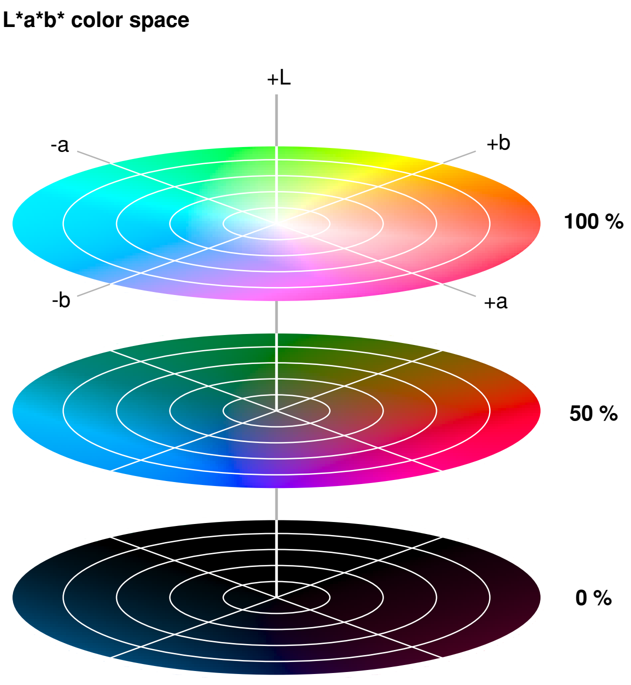


L*u*v* Space and L*a*b* Space

One problem with representing colors using the XYZ and Yxy color spaces is that they are perceptually nonlinear: it is not possible to accurately evaluate the perceptual closeness of colors based on their relative positions in XYZ or Yxy space. Colors that are close together in Yxy space may seem very different to observers, and colors that seem very similar to observers may be widely separated in Yxy space.

L*u*v* space and L*a*b* space are nonlinear transformations of the XYZ tristimulus space. These spaces are designed to have a more uniform correspondence between geometric distances and perceptual distances between colors that are seen under the same reference illuminant. A rendering of L*a*b space is shown in Figure 2-6 (page 22).

Figure 2-6 L*a*b* color space



Both L*u*v* space and L*a*b* space represent colors relative to a reference white point, which is a specific definition of what is considered white light, represented in terms of XYZ space, and usually based on the whitest light that can be generated by a given device.

Important: Because L*u*v* space and L*a*b* space represent colors relative to a specific definition of white light, they are not completely device independent; two numerically equal colors are truly identical only if they were measured relative to the same white point.

Measuring colors in relation to a white point allows for color measurement under a variety of illuminations.

A primary benefit of using L*u*v* space and L*a*b* space is that the perceived difference between any two colors is proportional to the geometric distance in the color space between their color values, if the color differences are small. Use of L*u*v* space or L*a*b* space is common in applications where closeness of color must be quantified, such as in colorimetry, gemstone evaluation, or dye matching.

Indexed Color Spaces

In situations where you use only a limited number of colors, it can be impractical or impossible to specify colors directly. If you have a bitmap with only a few bits per pixel (1, 2, 4, or 8, for example), each pixel is too small to contain a complete color specification; its color must be specified as an index into a list or table of color values. If you are using spot colors in printing or pen colors in plotting, it can be simpler and more precise to specify each color as an index into a list of colors instead of an actual color value. Also, if you want to restrict the user to drawing with a specific set of colors, you can put the colors in a list and specify them by index.

Indexed space is the color space you use when drawing with indirectly specified colors. An indexed color value (a color specification in indexed color space) consists of an index value that refers to a color in a color list. Color values are defined in Color-Component Values, Color Values, and Colors (page 23).

Named Color Spaces

In a named color space, each color has a name; colors are generally ordered so that each has an equal perceived distance from its neighbors in the color space. A named color space provides a relatively small number of discrete colors.

Color systems using named color spaces have existed for many years. Graphic artists and designers using named color systems can see the real color by looking at a color chip or swatch. Printing shops can reproduce a specified color accurately.

Named color systems are useful for spot colors, but they have several drawbacks:

- They are not useful for images, which require a continuous range of colors.
- They are highly device dependent and proprietary.
- Colors are tied to medium-specific formulations.
- Applications that use these systems require a device-specific database for each supported printer, making it difficult to add additional devices.

Color-Component Values, Color Values, and Colors

Each of the color spaces described here requires one or more numeric values in a particular format to specify a color.

Each dimension, or component, in a color space has a color-component value. An unsigned 16-bit color-component value can vary from 0 to 65,535 (0xFFFF), although the numerical interpretation of that range is different for different color spaces. In most cases, color-component intensities are interpreted numerically as varying between 0 and 1.0. An exception occurs for the a* and b* channels of the Lab color space, where values ranging from 0 to 65,535 are interpreted numerically as varying from -128.0 to approximately 128.0.

Depending on the color space, one, two, three, or four color-component values combine to make a color value. For HiFi colors, up to eight color-component values combine to make a color. A color value is a structure; it is the complete specification of a color in a given color space.

Color Conversion and Color Matching

Color conversion is the process of converting colors from one color space to another. Color matching, which entails color conversion, is the process of selecting colors from the destination gamut that most closely approximate the colors from the source image. Color matching always involves color conversion, whereas color conversion may not entail color matching. Rendering intent refers to the approach taken when a CMM maps or translates the colors of an image to the color gamut of a destination device—that is, a rendering intent specifies a gamut-matching strategy.

Different imaging devices (scanners, displays, printers) work in different color spaces and each is capable of producing a different range of colors. Although color displays from different manufacturers all use RGB colors, each will typically have a different RGB gamut. Printers that work in CMYK space vary drastically in their gamuts, especially if they use different printing technologies. Even a single printer's gamut can vary significantly with the ink or type of paper it uses. It's easy to see that conversion from RGB colors on an individual display to CMYK colors on an individual printer using a particular paper type can lead to unpredictable results.

When an image is output to a particular device, the device displays only those colors that are within its gamut. Likewise, when an image is created by scanning, all colors from the original image are reduced to the colors within the scanner's gamut. Devices with different gamuts cannot reproduce each other's colors exactly, but careful shifting of the colors used on one device can improve the visual match when the image is displayed on another.

Figure 2-7 Color gamuts for two devices expressed in Yxy space

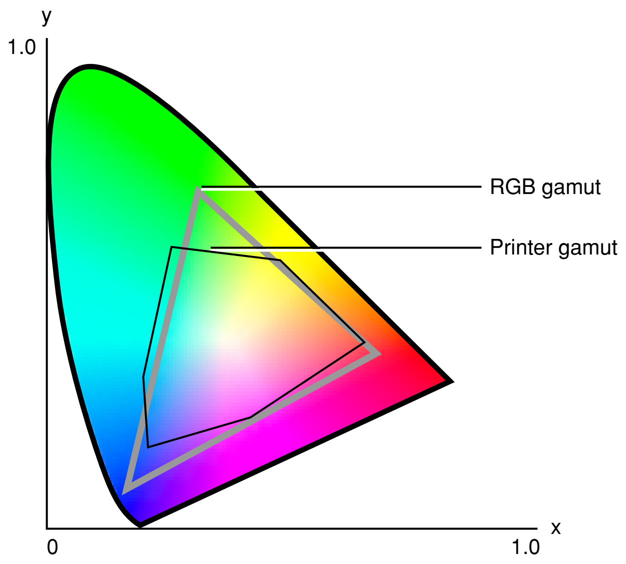


Figure 2-7 (page 24) shows examples of two devices' color gamuts, projected onto Yxy space. Both devices produce less than the total possible range of colors, and the printer gamut is restricted to a significantly smaller range than the RGB gamut. The problem illustrated by Figure 2-7 (page 24) is to display the same image on both devices with a minimum of visual mismatch. The solution to the problem is to match the colors of the image using profiles for both devices and one or more color management modules. A profile is a structure that provides a means of defining the color characteristics of a given device in a particular state. For more information, see Profiles (page 30).

Color Management Systems

Members of the computer and publishing industries have developed color management systems (CMSs) to convert colors from the color space of one device to the color space of another device (for example, from a scanner to a monitor). The components of a color management system include

- collections of color characteristics (these collections are given various names, such as color tags, precision transforms, or profiles)
- a color management module (CMM) that performs the color matching among, and transformation between, collections of color characteristics; for more information, see Color Management Modules (page 36)
- a programming interface for invoking color matching

The goal of these systems is to provide consistent color across peripheral devices and across operating-system platforms. Most CMSs are proprietary, but ColorSync, the platform-independent color management system from Apple Computer, supports the industry-standard color profile specification currently defined by the International Color Consortium (ICC). The ICC publishes the International Color Consortium Profile Format Specification. To obtain a copy of the specification, or to get other information about the ICC, visit the ICC Web site at <http://www.color.org/>.

A color management system gives the user the ability to perform color matching, to see in advance which colors cannot be accurately reproduced on a specific device, to simulate the range of colors of one device on another, and to calibrate peripheral devices using a device profile and a calibration application.

Overview of ColorSync

This section describes ColorSync and the ColorSync Manager. ColorSync is the platform-independent color management system from Apple Computer, Inc. ColorSync provides essential services for fast, consistent, and accurate color calibration, proofing, and reproduction. The ColorSync Manager is the application programming interface (API) to these services on the Mac OS.

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

If you are unfamiliar with terms and concepts such as CMM, profile, color space, and color management, or would like to review these topics, read Overview of Color and Color Management Systems (page 13) before reading this section.

What's New (page 151) lists the new features available with ColorSync 2.5 and provides links to new and revised material in this document. It also describes where to obtain documentation for other color-related technologies.

About ColorSync

ColorSync is the first system-level implementation of an industry-standard color management system. Even in a system in which all input, output, and display devices are new and perfectly calibrated, there will be differences in device gamuts (the ranges of color that can be reproduced by the devices). ColorSync can correct for such differences in device gamuts, as well as for differences caused by aging of filter sets and lamps on scanners, phosphor decay and ambient light on monitors, and differences in pigments and substrates on output devices such as printers and presses. As a result, it is possible to maintain accurate color across many possible input, display, and output devices.

Developers writing device drivers use the ColorSync Manager to support color matching between devices. Application developers use the ColorSync Manager to communicate with drivers and to present users with color-matching information, such as a device's color capabilities.

Why You Should Use ColorSync

Different imaging devices such as scanners, displays, and printers work in different color spaces, and each can have a different gamut (the range of colors a device can display). Color displays from different manufacturers all use RGB colors but may have different RGB gamuts as a result of the type and age of the phosphors used. Printers and presses work in CMYK space and can vary drastically in their gamuts, especially if they use different printing technologies. Even a single printer's gamut can vary significantly depending on the ink or type of paper in use. It's easy to see that conversion from RGB colors on an individual display to CMYK colors on an individual printer using a specific ink and paper type can lead to unpredictable results.

The ColorSync Manager addresses these problems by providing applications and color peripheral device drivers with device-independent color-matching and color conversion services based on the ColorSync color management system. With this ColorSync support, the user can quickly and accurately convert color images for optimal results on a specified device.

The ColorSync Advantage

There are many reasons you should consider adding ColorSync support to your products:

- Working with color is more difficult than many people realize, but ColorSync provides a highly effective system to help you perform accurate, industry-standard color management.
- Devices age and are subject to continuous inconsistence due to phosphor aging, pigment changes, substrate differences (white point, absorption, reflectivity), filter aging, and change in the life of the luminant.
- Artists, designers, and prepress experts need to achieve repeatable, reliable, and consistent color—onscreen, in print, and for electronic delivery to multimedia and the Internet. ColorSync is the tool of choice for meeting these requirements.
- There are many color measurement instruments, an array of profiling software, and a wide selection of production tools—page make-up, image editing, illustration, image database, Photo CD applications, and more—that already support ColorSync.
- ColorSync is widely available to Macintosh users, who make up the large majority of graphic arts and publishing professionals.
- Starting with version 2.5, the ColorSync Manager provides an extensible AppleScript framework that allows users to script many common tasks, such as matching an image or embedding a profile in an image. These AppleScript capabilities, described in Scripting Support (page 43), make it possible to automate many workflow processes.

Color Management in Action

Firms that sell clothing through mail order catalogs and Web sites report the most common reason a customer returns an item is color: when the product arrived, it didn't match the color in the catalog or on the monitor. Working with managed color, however, these firms can ensure the original image is scanned accurately, the catalog or monitor displays the color correctly, the output device prints the color correctly, and the customers see the color as accurately as they are able.

ColorSync Manager Overview

This section provides an overview of the ColorSync Manager and how an application or device driver can use it for color conversion, color matching, color gamut checking, profile management, monitor calibration, and creating color management modules (CMMs).

The ColorSync Manager provides a set of routines contained in a system extension. ColorSync also includes a collection of display device profiles for all Apple color monitors, some default profiles for standard color spaces, and a robust default CMM. In addition, the ColorSync control panel, shown in Figure 3-1 (page 33), allows a user to specify a preferred CMM and various default profiles. CMMs and profiles are discussed throughout this material.

To provide its color-matching services, the ColorSync Manager works with color profiles and with one or more color management modules. Your application or driver can supply its own CMM, or you can use the default CMM, a robust CMM that is installed as part of the ColorSync extension and supports all the required and optional functions defined by the ColorSync Manager. The ColorSync Manager relies on the Component Manager to support plug-and-play capability for third-party CMMs. The Component Manager is described in Inside Macintosh: More Macintosh Toolbox. For more information on CMMs, see Color Management Modules (page 36).

A profile is a table that contains the color characteristics of a given device in a particular state. ColorSync profiles conform to the format currently defined by the International Color Consortium (ICC). Device driver developers and peripheral manufacturers can provide their own profiles or they can obtain profiles from a number of vendors. For a list of profile vendors, see the ColorSync Web site at <http://colorsync.apple.com/>. Profiles are described in detail in Profiles (page 30).

ColorSync Versions

This document covers the ColorSync Manager through version 2.5. Most existing code written to use version 2.0 or 2.1 of the ColorSync Manager should continue to work with version 2.5 without modification.

This document uses a specific version number, such as version 2.5 of the ColorSync Manager, only where necessary to identify features associated with a particular version. It may use 2.x to refer inclusively to ColorSync versions 2.0, 2.1, and 2.5, or to refer to the profile format used with these versions—the meaning should be clear from the context. For more information on profile version numbers, see ColorSync and ICC Profile Format Version Numbers (page 30).

For a description of the `Gestalt` information, shared library version numbers, CMM version numbers, and ColorSync header files you use with different versions of the ColorSync Manager, see ColorSync Version Information (page 141).

Minimum Requirements For Running ColorSync 2.5

ColorSync version 2.5 requires Mac OS version 7.6.1 or newer, running on a Power Macintosh or on a 68K Macintosh computer with a 68020 or greater processor. For information on system and CPU requirements for previous ColorSync versions, and for related information such as Gestalt, shared library, and CMM versions, see ColorSync Version Information (page 141).

Programming Interface

The ColorSync Manager application programming interface (API) allows your application or device driver to handle such tasks as color matching, color conversion, profile management, and profile searching. You can access and read individual tagged elements within a profile, embed profiles in documents, modify profiles, and transform images through various CMMs to perform color matching and profile data transfer from one format to another.

The ColorSync API is summarized in Summary of the ColorSync Manager (page 107). You can find detailed information about individual functions, constants, and data types in *ColorSync Manager Reference*. For code samples that show how to use the ColorSync API, see Developing ColorSync-Supportive Applications (page 49). For information on the ColorSync Manager header files, see ColorSync Header Files (page 143).

The ColorSync Manager is implemented as a shared library on PowerPC-based computers. For a listing of the shared library version number for each released version of the ColorSync Manager, along with corresponding `Gestalt` selector codes, see Gestalt, Shared Library, and CMM Version Information (page 142).

Profiles

To perform color matching or color conversion across different color spaces requires the use of a profile for each device involved. Profiles provide the information necessary to understand how a particular device reproduces color. A profile may contain such information as lightest and darkest possible tones (referred to as white point and black point), the difference between specific targets and what is actually captured, and maximum densities for red, green, blue, cyan, magenta, and yellow. Together these measurements represent the data which describe a particular color gamut.

The International Color Consortium Profile Format

ColorSync supports the profile format defined by the International Color Consortium (ICC). The ICC format provides a single cross-platform standard for translating color data across devices. The ICC defines several types of profiles, including input, output, and display profiles. Each of these types specifies a different required set of information, but all follow the same format.

The founding members of the ICC include Adobe Systems Inc.; Agfa-Gevaert N.V.; Apple Computer, Inc.; Eastman Kodak Company; FOGRA (Honorary); Microsoft Corporation; Silicon Graphics, Inc.; and Sun Microsystems, Inc. These companies have committed to full support of this specification in their operating systems, platforms, and applications.

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/>.

ColorSync and ICC Profile Format Version Numbers

The first version of ColorSync used a 1.0 profile format that preceded the ICC profile format definition. Starting with version 2.0 of the ColorSync Manager, ColorSync uses a 2.x profile format that supports all current ICC profile format versions. As of this writing, that includes ICC versions 2.0 and 2.1. The ICC defines the profile format version as part of the profile header. For more information on the differences between these profile format versions, see ColorSync 1.0 Profiles and Version 2.x Profiles (page 145).

Source and Destination Profiles

When a ColorSync-supportive scanning application creates a scanned image, it embeds a profile for the scanner in the image. The profile that is associated with the image and describes the characteristics of the device on which the image was created is called the source profile. If the colors in the image are subsequently converted to another color space by the scanning application or by another ColorSync-supportive application, ColorSync can use that source profile to identify the original colors and to match them to colors expressed in the new color space.

Displaying the image requires using another profile, which is associated with the output device, such as a display. The profile for that device is called the destination profile. If the image is destined for a display, ColorSync can use the display's profile (the destination profile) along with the image's source profile to match the image's colors to the display's gamut. If the image is printed, ColorSync can use the printer's profile to match the image's colors to the printer, including generating black and removing excessive color densities (known as undercolor removal, or UCR) where appropriate.

Profile Classes

The ColorSync Manager supports seven classes, or types, of profiles. These classes are defined below. Three of the profile classes define device profiles for different types of devices: input, output, and display devices. The other four profile classes include definitions for an abstract profile, a color space profile, a named color space profile, and a device link profile. The constants used to specify these classes are described in *ColorSync Manager Reference*.

A device profile characterizes a particular device: that is, it describes the characteristics of a color space for a physical device in a particular state. A display, for example, might have a single profile, or it might have several, based on differences in gamma value and white point. A printer might have a different profile for each paper type or ink type it uses because each paper type and ink type constitutes a different printer state. When an application calls a ColorSync Manager color-matching function to match colors between devices, such as a display and a printer, it specifies the profile for each device.

Device profiles are divided into three broad classifications:

- input devices, such as scanners and digital cameras
- display devices, such as monitors and flat-panel screens
- output devices, such as printers, film recorders, and printing presses.

Each device profile class has its own signature. The ColorSync constants for these signatures are described in *ColorSync Manager Reference*. For related information, see Devices and Their Profiles (page 109).

A profile connection space (PCS) is a device-independent color space used as an intermediate when converting from one device-dependent color space to another. Profile connection spaces are typically based on spaces derived from the CIE color space, which is described in Device-Independent Color Spaces (page 19). ColorSync supports two of these spaces, XYZ and L*a*b.

A color space profile contains the data necessary to convert color values between a PCS and a non-device color space (such as L*a*b to or from L*u*v, or XYZ to or from Yxy), for color matching. The ColorSync Manager uses color space profiles when mapping colors between different color spaces. Color space profiles also provide a convenient means for CMMs to convert between different non-device profiles.

L*a*b, L*u*v, XYZ, and Yxy color spaces are described in Color Spaces (page 15).

Abstract profiles allow applications to perform special color effects independent of the devices on which the effects are rendered. For example, an application may choose to implement an abstract profile that increases yellow hue on all devices. Abstract profiles allow users of the application to make subjective color changes to images or graphics objects.

A device link profile represents a one-way link or connection between devices. It can be created from a set of multiple profiles, such as various device profiles associated with the creation and editing of an image. It does not represent any device model, nor can it be embedded into images.

For more information on device link profiles, see `CWNewLinkProfile` and `CMConcatProfileSet`.

Important: The `CMConcatProfileSet` structure used to create a device link profile includes a field that identifies the one CMM to use for the entire color-matching session across all profiles. However, you should read How the ColorSync Manager Selects a CMM (page 51), for a full description of the algorithm ColorSync uses to choose a CMM.

A named color space profile contains data for a list of named colors. The profile specifies a device color value and the corresponding CIE value for each color in the list. Profiles are typically stored as individual files in the ColorSync Profiles folder. For example, device-specific profiles provided by hardware vendors should be stored in the ColorSync Profiles folder. The location and use of the ColorSync Profiles folder has changed beginning in version 2.5. For a description of these changes, see Profile Search Locations (page 34).

Profiles can also be embedded within images. For example, profiles can be embedded in PICT, EPS, and TIFF files and in the private file formats used by applications. Embedded profiles allow for the automatic interpretation of color information as the color image is transferred from one device to another.

Note: The ICC profile format implemented in a ColorSync version 2.x profile is significantly different from the ColorSync 1.0 profile implementation. For more information, see ColorSync and ICC Profile Format Version Numbers (page 30). <8bat>u

Embedding a profile in an image guarantees that the image can be rendered correctly on a different system. However, profiles can be large—the largest can be several hundred KB or even larger. A profile identifier is an abbreviated data structure that uniquely identifies, and possibly modifies, a profile in memory or on disk, but takes up much less space than a large profile. For example, an application might embed a profile identifier to change just the rendering intent in an image without having to embed an entire new profile. Rendering intents are described in Rendering Intents (page 37). For more information on embedding profile information, see Embedding Profiles and Profile Identifiers (page 69).

Important: A document containing an embedded profile identifier (as opposed to an embedded profile) is not necessarily portable to different systems or platforms.

Profile Properties

Profiles can contain different kinds of information. For example, a scanner profile and a printer profile have different sets of minimum required tags and element data. However, all profiles have at least a header followed by a required element tag table. The required tags may represent lookup tables, for example. The required tags for various profile classes are described in the International Color Consortium Profile Format Specification.

Profiles contain additional information, such as a specification for how to apply matching. For more information, see Color Management Modules (page 36). Profiles may also have a series of optional and private tagged elements. These private tagged elements may contain custom information used by particular color management modules.

Profile Location

In most cases, a ColorSync version 2.x profile is stored in a disk file. However, to support special requirements, a profile can also be located in memory or in an arbitrary location that is accessed by a procedure you specify. See *ColorSync Manager Reference* for a description of ColorSync Manager structures for working with profiles that are stored in each of these locations. See Opening a Profile and Obtaining a Reference to It (page 58) for information on working with profile locations in your application.

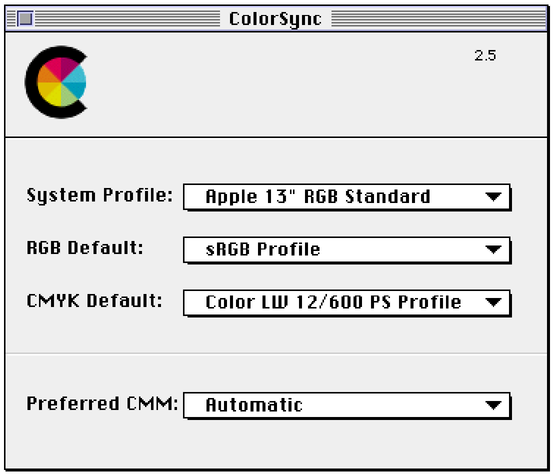
Setting Default Profiles

Prior to version 2.5, the default system profile (or simply the system profile) served as the default display profile; it also served as the default profile for color operations for which no profile was specified. The system profile had to be an RGB profile. A user could specify the system profile through the ColorSync control panel (formerly called the ColorSync<Superscript>™ System Profile control panel). If a user did not specify a system profile, then by default ColorSync used the Apple 13-inch color display profile.

Because the system profile was used for two dissimilar functions (default display profile and default profile for some RGB operations), there were several limitations:

- A user could not specify default profiles for color spaces other than RGB.
- A user could not specify separate profiles for more than one monitor.
- When matching an image without an embedded monitor profile, no matching occurred because the source and destination profiles were the same (system) profile.

Figure 3-1 The ColorSync control panel



System Profile: Apple 13" RGB Standard
RGB Default: sRGB Profile
CMYK Default: Color LW 12/600 PS Profile
Preferred CMM: Automatic

Starting with ColorSync version 2.5, a user can set default profiles for RGB and CMYK color spaces, as well as for the system profile, using the ColorSync control panel shown in Figure 3-1 (page 33). In addition, your application can call routines to get and set default profiles for the RGB, CMYK, Lab, and XYZ color spaces. As in previous versions of ColorSync, you can also call routines to get and set the current system profile.

Also starting with ColorSync version 2.5, a user can specify a separate profile for each monitor using the Monitors & Sound control panel. In addition, your application can call routines to get and set the profile for each display.

Important: When a user sets a profile for a monitor in the Monitors & Sound control panel, ColorSync makes that profile the current system profile. When your application sets a profile for a monitor, it may also wish to make that profile the system profile.

Because ColorSync version 2.5 provides capabilities for getting and setting default profiles for color spaces and for assigning a profile to each monitor, your application and anyone using it can more precisely specify source and destination profiles. For example, your application can set the destination profile for an operation to be the profile for a specific monitor or the source profile to be the default CMYK profile.

Important: Under certain conditions, functions such as `NCMUseProfileComment` still use the system profile, so you should set the system profile to an appropriate value, such as a profile for your main display.

For information on getting and setting profiles in code, see *ColorSync Manager Reference*. Monitor Calibration and Profiles (page 41), describes how a user can specify a separate profile for each available monitor.

Profile Search Locations

ColorSync uses the ColorSync Profiles folder as a common location for profile files. When you install ColorSync, for example, it puts a number of default monitor profiles in the ColorSync Profiles folder. Users should also store custom profiles there, and ColorSync functions that search for profiles begin their search in the profiles folder.

Starting with ColorSync version 2.5, the ColorSync Profiles folder is located in the System folder; in earlier versions the folder was named ColorSync<Superscript>™ Profiles and was located in the Preferences folder. The new location protects profile files from deletion when the Preferences folder is deleted. More importantly, placement in the System folder will allow the profiles folder to become a magic folder, providing the following benefits:

- Starting with version 8.5 of the Mac OS, profiles dragged onto the System folder are automatically routed to the profiles folder.
- Starting with version 8.5 of the Mac OS, ColorSync can use the Toolbox `FindFolder` routine to find the profiles folder, using the Folder Manager constant `kColorSyncProfilesFolderType`.

  Important: Your application should continue to call ColorSync's `CMGetColorSyncFolderSpec` function to obtain the location of the profiles folder—it should not use a hard-coded path to a specific folder.

For backward compatibility, ColorSync automatically inserts into the new profiles folder an alias to the old location (inside the Preferences folder), if that folder exists and contains any profiles.

Where ColorSync Searches for Profiles

Prior to ColorSync 2.5, profile search routines such as `CMNewProfileSearch` looked for profiles only in the profiles folder (within the Preferences folder). Starting with version 2.5, the search routines look in the following locations:

- in the ColorSync Profiles folder (within the System folder)

- in first-level subfolders of the ColorSync Profiles folder
- in locations specified by aliases in the ColorSync Profiles folder (whether the aliases are to single profiles or to folders containing profiles)

With this searching support, you can group profiles in subfolders within the profiles folder (one level of subfolders is currently allowed). For example, you might store all scanner profiles in one folder and a variety of monitor profiles for your primary monitor in another. You can also store aliases to other profiles and profile folders within the ColorSync Profiles folder. ColorSync search routines will find all profiles in the specified locations.

The Profile Cache and Optimized Searching (page 35) describes how your application can perform optimized profile searching with ColorSync 2.5.

Where ColorSync Does Not Look for Profiles

Because profile searching in ColorSync 2.5 can only go two levels deep, ColorSync search routines will not find a profile in the following cases:

- The profile is located in a folder that is within a folder in the profiles folder (requires more than two levels of searching).
- The profile is located in a folder that is within a folder specified by an alias in the profiles folder (again, requires more than two levels of searching).
- The profile is in a folder whose name starts with a parenthesis.
- The profile is specified by an alias to an unmounted volume (only mounted volumes are searched).

Temporarily Hiding a Profile Folder

To temporarily hide a folder from ColorSync's search path, put parentheses around the name of the folder or the alias to the folder.

The Profile Cache and Optimized Searching

Starting with version 2.5, ColorSync creates a cache file (containing private data) in the Preferences folder to keep track of all currently-installed profiles. The cache stores key information about each profile, using a smart algorithm that avoids rebuilding the cache unless the profile folder has changed.

ColorSync takes advantage of the profile cache to speed up profile searching. This optimized searching can help your application speed up some operations, such as displaying a pop-up menu of available profiles.

ColorSync's intelligent cache scheme provides the following advantages in profile management:

- The cache contains information including the name, header, script code, and location for each installed profile, so that once the cache has been built, ColorSync can supply the information your application needs for many tasks without having to reopen any profiles.
- When you call a search routine, ColorSync can quickly determine if there has been any change to the currently-installed profiles. If not, ColorSync can supply information from the cache immediately, giving the user a pleasing performance experience.

ColorSync 2.5 provides a flexible new routine, `CMIterateColorSyncFolder`, that takes full advantage of the profile cache to provide truly optimized searching and quick access to profile information. For an example of how to use this routine in your application, see Performing Optimized Profile Searching (page 81).

Important: Your application should use the `CMIterateColorSyncFolder` function, or one of the other ColorSync search functions described in *ColorSync Manager Reference*, to search for a profile, even if you are only looking for one file. Do not search for a profile by obtaining the location of the profiles folder and searching for the file directly.

Note that calls to the ColorSync search routines available before version 2.5 cannot take full advantage of the profile cache. For example, with the `CMNewProfileSearch` routine, the caller passes in a search criteria and gets back a list of profiles that match that criteria. Before version 2.5, ColorSync had to open each profile to build the list, and the caller was likely to open each profile again after getting the list back. With version 2.5, ColorSync can at least use the profile cache to narrow down the list (unless the search criteria asks for all profiles!), but it cannot fully optimize the search process.

Color Management Modules

A color management module (CMM) is a component that implements color-matching, color conversion, and gamut-checking services. A 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.

Each profile header includes a field that specifies a CMM to use for performing color matching involving that profile. If two profiles in a color-matching session specify different CMMs, or if a specified CMM is unavailable or unable to perform a requested function, the ColorSync Manager follows an algorithm, described in How the ColorSync Manager Selects a CMM (page 51), to determine which CMM to use.

ColorSync ships with a robust CMM that is installed as part of the ColorSync extension. This CMM supports all the required and optional functions defined by the ColorSync Manager, so it can always be used as a default CMM when another CMM is unavailable or unable to perform an operation.

The ColorSync Manager includes color conversion functions that allow your application or driver to convert colors between color spaces belonging to the same base families without the use of CMMs; CMMs themselves can also call these color conversion functions. However, color conversion and color matching across color spaces belonging to different base families always entail the use of a CMM.

When colors from one device's gamut are displayed on a device with a different gamut, as shown in Figure 2-7 (page 24), ColorSync can minimize the perceived differences in the displayed colors between the two devices by mapping the out-of-gamut colors into the range of colors that can by produced by the destination device.

A CMM uses lookup tables and algorithms for color matching, previewing color reproduction capabilities of one device on another, and checking for out-of-gamut colors (colors that cannot be reproduced). Although ColorSync provides a default CMM, device manufacturers and peripheral developers can create their own CMMs, tailored to the specific requirements of their device. For information on creating CMMs, see Developing Color Management Modules (page 121) and *ColorSync Manager Reference*. ColorSync also provides the Kodak CMM as a custom install feature, for users who wish to work with that CMM.

Setting a Preferred CMM

Starting with version 2.5, the ColorSync control panel, shown in Figure 3-1 (page 33), lets you choose a preferred CMM from any CMMs that are present (that is, registered with the Component Manager—see Creating a Component Resource for a CMM (page 123) for a description of how to create a CMM the ColorSync Manager can use).

If you choose a preferred CMM with the ColorSync control panel, and if that CMM is available, ColorSync will attempt to use that CMM for all color conversion and matching operations. If you specify Automatic instead, or if the specified CMM is no longer present or cannot provide the required matching service, or for versions prior to version 2.5, ColorSync follows an algorithm described in How the ColorSync Manager Selects a CMM (page 51) to determine which available CMM to use for matching.

Note that if you want color conversion and matching operations to use the same CMM-selection algorithm they did in versions prior to ColorSync 2.5, specify Automatic in the ColorSync control panel. <8bat>u

Starting with ColorSync 2.5, your application can determine the preferred CMM by calling the function `CMGetPreferredCMM`.

Rendering Intents

Rendering intent refers to the approach taken when a CMM maps or translates the colors of an image to the color gamut of a destination device—that is, a rendering intent specifies a gamut-matching strategy. The ICC specification defines a profile tag for each of four rendering intents: perceptual matching, relative colorimetric matching, saturation matching, and absolute colorimetric matching. These rendering intents are described in Table 3-1.

Table 3-1 ICC rendering intents and typical image content

ICC term	Description	Typical content
perceptual matching	All the colors of one gamut are scaled to fit within another gamut. Colors maintain their relative positions. Usually produces better results than colorimetric matching for realistic images such as scanned photographs. The eye can compensate for gamuts differences, such as in Figure 2-7 (page 24), and when printed on a CMYK device, the image may look similar to the original on an RGB device. A side effect is that most of the colors of the original space may be altered to fit in the new space.	photographic
relative colorimetric matching	Colors that fall within the overlapping gamuts of both devices are left unchanged. For example, to match an image from the RGB gamut onto the CMYK printer gamut in Figure 2-7 (page 24), only the colors in the RGB gamut that fall outside the printer gamut are altered. Allows some colors in both images to be exactly the same, which is useful when colors must match quantitatively. A disadvantage is that many colors may map to a single color, resulting in tone compression. All colors outside the printer gamut, for example, would be converted to colors at the edge of its gamut, reducing the number of colors in the image and possibly altering its appearance. Colors outside the gamut are usually converted to colors with the same lightness, but different saturation, at the edge of the gamut. The final may be lighter or darker overall than the original image, but the blank areas will coincide.	spot colors

ICC term	Description	Typical content
saturation matching	The relative saturation of colors is maintained as well as can be achieved from gamut to gamut. Colors outside the gamut of the to space are usually converted to colors with the same saturation of the from space, but with different lightness, at the edge of the gamut. Can be useful for some graphic images, such as bar graphs and pie charts, when the actual color displayed is less important than its vividness.	business graphics
absolute colorimetric matching	Preserves native device white point of source image instead of mapping to D50 relative. Most often using in simulation or proofing operations where a device is trying to simulate the behavior of another device and media. For example, simulating newsprint on a monitor with absolute colorimetric intent would allow white space to be displayed onscreen as yellowish background because of the differences in white points between the two devices.	no typical content (most often used in proofing)

Color professionals and technically-sophisticated users are likely to be familiar with the ICC terms for rendering intent and the gamut-matching strategies they represent. If your application is aimed at novice users, however, you may prefer to add a simplified terminology based on the typical image content associated with a rendering intent (for example, perceptual matching is commonly used for photographic images). Table 3-1 lists the ICC-specified rendering intents and the corresponding image content term, where applicable. Note that there is no simplified terminology for describing absolute colorimetric matching. However, novice users are not likely to need this rendering intent.

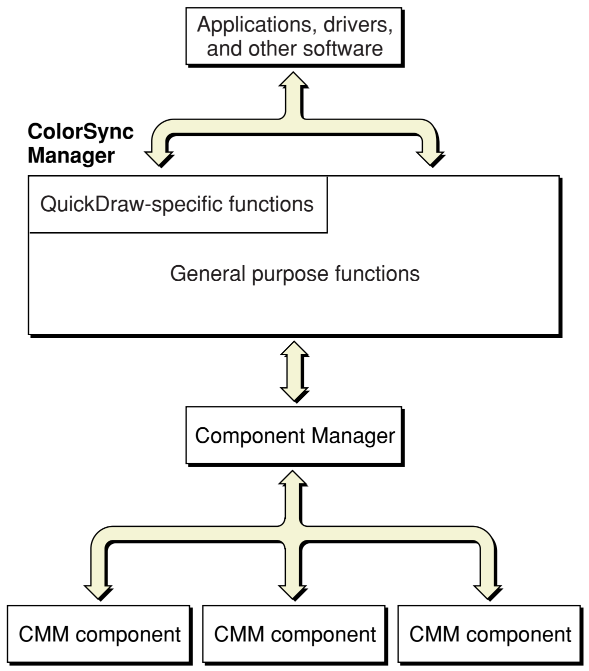
For information about the actual rendering intent tags, you can obtain the latest ICC profile specification by visiting either the ICC Web site at <http://www.color.org/> or the ColorSync Web site at <http://colorsync.apple.com/>.

When Color Matching Occurs

When the color gamut of a source profile is different from the color gamut of a destination profile, ColorSync relies on the CMM and the information stored in both profiles for mapping the colors from the source profile's gamut to the destination profile's gamut. The CMM contains the necessary algorithms and lookup tables to enable consistent color mapping among devices.

When an application or device driver uses the ColorSync Manager functions for color matching, it specifies the source and destination profiles. If it does not specify the source profile or the destination profile for a matching operation, ColorSync substitutes a default profile. See Setting Default Profiles (page 33) for more information.

Figure 3-2 The ColorSync Manager and the Component Manager


Applications, drivers, and other software
ColorSync Manager
QuickDraw-specific functions
General purpose functions
Component Manager
CMM component
CMM component
CMM component

Color matching between the source and destination color spaces happens inside the color management module (CMM) component. Figure 3-2 (page 39) shows the relationship between your application or device driver, the ColorSync Manager, the Component Manager, and one or more available CMM components.

Your application can call any ColorSync Manager function, whether QuickDraw-specific or general purpose. One of three things then happens:

- The ColorSync Manager routine performs the operation directly.
- The ColorSync Manager communicates with a CMM through the Component Manager.
- The ColorSync Manager calls other ColorSync routines that communicate with a CMM through the Component Manager.

General purpose and QuickDraw-specific functions are described in the following sections.

General Purpose Color-Matching Functions

A general purpose color-matching function is one that uses a color world to characterize how to perform color-matching. General purpose functions depend on the information contained in the profiles that you supply when you set up the color world. You can define a color world for color transformations between a source profile and a destination profile, or define a color world for color transformations between a series of concatenated profiles.

Creating a Color World to Use With the General Purpose Functions (page 64) provides a code sample for working with general purpose functions. Matching Colors Using the General Purpose Functions (page 66) lists the general purpose functions and provides a description of each function.

In contrast to the general purpose color-matching functions, the QuickDraw-Specific Color-Matching Functions (page 40) are tailored for color-matching with QuickDraw. Note, however, that you can also use the general purpose functions when working with QuickDraw—for example, if you need the greater level of control the general purpose functions provide.

QuickDraw-Specific Color-Matching Functions

A QuickDraw-specific color-matching function is one that uses QuickDraw to provide images showing consistent colors across displays. The ColorSync Manager provides two QuickDraw-specific functions that your application can call to draw a color picture to the current display:

- `NCMBeginMatching` uses the source and destination profiles you specify to match the colors of the source image to the colors of the device for which it is destined.
- `NCMDrawMatchedPicture` matches a QuickDraw picture's colors to a destination device's color gamut as the picture is drawn, using the specified destination profile. Uses the system profile as the initial source profile but switches to any embedded profiles as they are encountered.

Matching to Displays Using QuickDraw-Specific Operations (page 62) provides a code sample for working with QuickDraw-specific functions. *ColorSync Manager Reference* lists the QuickDraw-specific functions and provides a description of each function. Note that the QuickDraw-specific functions call upon the general purpose functions to perform their operations, as shown in Figure 3-2 (page 39).

Converting Between Color Spaces

Color conversion, which does not require the use of color profiles, is a much simpler process than color matching. The ColorSync Manager provides functions your application can call to convert a list of colors within the same base family—that is, between a base color space and any of its derived color spaces or between two derivatives of the same base family.

You can convert a list of colors between XYZ and any of its derived color spaces, which include L*a*b*, L*u*v*, and Yxy, or between any two of the derived color spaces. You can also 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.

You can convert a list of colors between RGB, which is the base-additive device-dependent color space, and any of its derived color spaces, such as HLS, HSV, and Gray, or between any two of the derived color spaces.

Note: The color conversion functions do not support conversion of HiFi colors. <8bat>u

Here are brief descriptions of the XYZ color space and its derivative color spaces:

- The XYZ space, referred to as the interchange color space, is the fundamental, or base CIE-based independent color space.
- The L*a*b* color space is a CIE-based independent color space used for representing subtractive systems, where light is absorbed by colorants such as inks and dyes. The L*a*b* color space is derived from the XYZ color space. The default white point for the L*a*b* interchange space is the D50 white point.
- The L*u*v* color space is a CIE-based color space used for representing additive color systems, including color lights and emissive phosphor displays. The L*u*v* color space is derived from the XYZ color space.

- The Yxy color space expresses the XYZ values in terms of x and y chromaticity coordinates, somewhat analogous to the hue and saturation coordinates of HSV space. This allows color variation in Yxy space to be plotted on a two-dimensional diagram.
- The XYZ color space includes two XYZ data type formats. The `CMFixedXYZColor` data type uses the Fixed data type for each of the three components. Fixed is a signed 32-bit value. The `CMFixedXYZColor` data type is also used in the ColorSync Manager 2.x profile header `CM2Header`. The `CMXYZColor` data type uses 16-bit values for each component.

Here are brief descriptions of the RGB color space and its derivative color spaces:

- The RGB color space is a three-dimensional color space whose components are the red, green, and blue intensities that make up a given color.
- The HLS and HSV color spaces belong to the family of RGB-based color spaces, which are directly supported by most color displays and scanners.
- Gray spaces typically have a single component, ranging from black to white. The Gray color space is used for black-and-white and grayscale display and printing.

To convert colors from one color space to one of its derived spaces, you don't need to specify source and destination profiles. Instead, you just call the appropriate ColorSync Manager function to convert between the desired color spaces. In cases where you're converting between XYZ and Lab or Luv spaces, a white reference is required to perform the conversion. This reference is expressed in the XYZ color space, and provides a way to specify the theoretical illuminant (for example, D65) under which the colors are viewed.

Note: Prior to version 2.1, the ColorSync Manager used a component to implement color conversion. An application had to open a connection to the component with the Component Manager, then pass the component instance as a parameter to the color conversion functions. For example, the `CMXYZToLab` function performs the same conversion as `CMConvertXYZToLab`, but takes a first parameter of

```
ComponentInstance ci
```

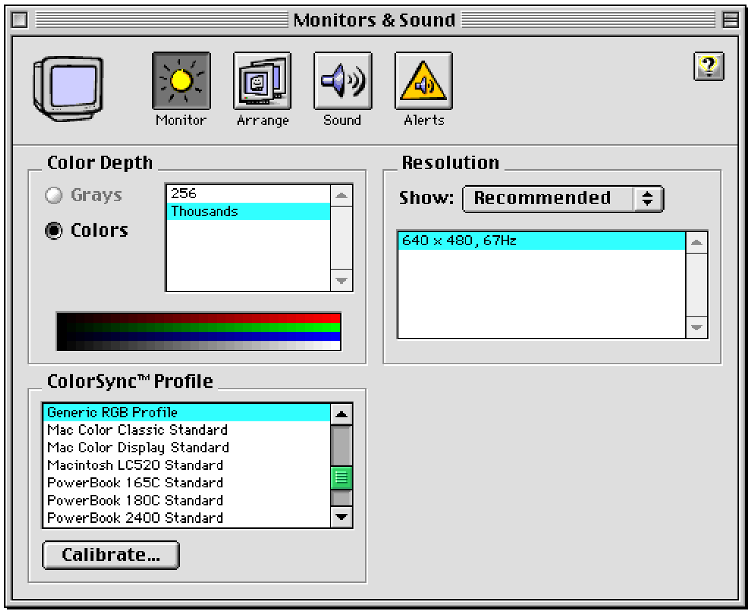
For backward compatibility, component-based color conversion functions such as `CMXYZToLab` are still supported. However, their use is discouraged, and they are not guaranteed to work in future versions (nor are they documented here).

Monitor Calibration and Profiles

Since ColorSync was first introduced, a common question from end users has been Where is the ColorSync profile for my monitor? The answer is that because some monitor manufacturers do not supply ColorSync profiles for their products, purchasers of third-party monitors may not have access to a profile that is specific to their monitor. As a result, they are unable to use ColorSync effectively.

Even when a user has a factory-supplied profile, switching to a different monitor setup can reduce the profile's accuracy. For example, if the user changes the monitor's gamma value and white point, the original profile is no longer useful. The user needs to run a calibration application (if one is available) to generate a new ColorSync profile for the new monitor settings.

Figure 3-3 Monitors & Sound Control Panel for ColorSync 2.5



Starting with version 2.5, ColorSync uses the Monitors & Sound control panel to provide a monitor calibration framework to help users obtain the monitor profiles they need. Figure 3-3 (page 42) shows the new Monitors & Sound control panel. Note that the list of gamma values (Mac Standard Gamma, uncorrected gamma) has been removed because that function is now part of the calibration process.

Note: If you develop a utility that adjusts gamma or modifies other calibration values, you should modify your software so that it uses the monitor calibration framework to gain system-level support.

Setting a Profile for Each Monitor

Because Monitors & Sound displays a panel for each available monitor, a user can also select, for each monitor, a separate profile from the list of available profiles. When a user sets a profile for a monitor in the Monitors & Sound control panel, ColorSync makes that profile the current system profile, as described in Setting Default Profiles (page 33). When your application sets a profile for a monitor, it may also wish to make that profile the system profile. If so, it must set the system profile explicitly by calling `CMSetSystemProfile`. For information on how to set monitor profiles in your code, see *ColorSync Manager Reference*.

Calibration

Calibration and characterization are related terms, but with important differences. Calibration is the process of setting a device's parameters according to its factory standards. This is also known as linearizing or linearization. Characterization is the process of learning the color character of a monitor so that a profile can be created to describe it. Unlike input and output devices, whose calibration and characterization steps are not the same, a monitor is calibrated and characterized in one step.

The Calibrate button on the Monitors & Sound control panel provides the launching point for monitor calibration. A user can calibrate each monitor and create one or more color profiles for each, based on variations in gamma, white point, and so on. For related information on profiles, see Profiles (page 30) and Devices and Their Profiles (page 109).

AppleVision and Apple ColorSync monitors are self-calibrating, so you will not see a Calibrate button for these monitors, unless there is a third-party calibrator installed in your Extensions folder.

Calibrating a monitor can be a challenging task for a naive user, but Apple Computer supplies a default calibrator that leads the user through a series of calibration steps. Using the default calibrator, even a novice should have a reasonable chance for success.

Note: There are limits to the effectiveness of monitor calibration by users. For example, some monitors, due to age or condition, cannot be calibrated, and a small percentage of the user population is color-blind.

The calibration framework uses a plug-in architecture that is fully accessible to third-party calibration plug-ins. When a user clicks on the Calibrate button, the Monitors & Sound control panel provides a list of all available calibrator plug-ins. To appear in the list, a plug-in must meet the following criteria:

- It must be stored in the Extensions folder.
- It must be a shared library (file type `'shlb'`).
- Its shared library must export the symbols `CanCalibrate` and `Calibrate`.
- It should have a unique creator type (registered with Apple).
- The name of the library's code fragment (specified in the `'cfrg'` resource) must be unique (among all currently loaded shared libraries) and begin with `'Cali'`. For example, you might want to name the library by appending your creator type to `'Cali'`.

You can find source code for sample monitor calibration plug-ins in the ColorSync 2.5.1 SDK. If you plan to create a monitor calibration plug-in, you should read the next section Video Card Gamma (page 43).

Video Card Gamma

Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma, which you specify with the `cmVideoCardGammaTag` constant. 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. When you call the function `CMSetProfileByAVID` and specify a profile that contains a video card gamma tag, ColorSync will extract the tag from the profile and set the video card based on the tag.

Important: The function `CMSetSystemProfile` does not retrieve video card gamma data to set the video card. Only the `CMSetProfileByAVID` function currently sets video card gamma data.

If you provide monitor calibration software, you should include the video card gamma tag in the profiles you create. For information on the constants and data types you use to work with video card gamma, see *ColorSync Manager Reference*. You can get more information about AVID values from the Display Manager SDK.

Scripting Support

Starting with version 2.5, the ColorSync Manager provides AppleScript support that allows users to script many common color-matching tasks. To provide this support, ColorSync now runs as a faceless background application (one with no user interface), rather than as a standard extension. By running as a background application, ColorSync can avoid namespace collisions and time-outs during long operations, and it can have its own AppleScript dictionary.

Note: You can examine ColorSync's full AppleScript dictionary by dragging the file ColorSync Extension from your Extensions folder onto the Script Editor application (usually located in the AppleScript folder within the Apple Extras folder).

Scriptable Properties

ColorSync provides scriptable support for getting and setting the following properties:

- system profile (the default system profile)
- default profiles for RGB, CMYK, Lab, and XYZ color spaces
- quit delay (the time in seconds for auto-quit, where 0 = never)
- profile location (a file specification)

For the following, you can only get, not set, the property:

- profile folder (the ColorSync profile folder)

Location is the only property currently supported for profiles, but future support is planned for additional profile properties.

Scriptable Operations

ColorSync supports the following scriptable operations:

- Matching an image.
- Matching an image with a device link profile.
- Proofing an image or a series of images.
- Embedding a profile in an image (very helpful for converting archives of legacy film).

Scriptable image operations currently work only on TIFF files, but support for other formats is planned.

Extending the Scripting Framework

The scripting framework uses a plug-in architecture that is fully accessible to third-party scripting plug-ins. When a user invokes a script to perform a ColorSync operation on an image, ColorSync (operating as a faceless background application) automatically builds a list of all available scripting plug-ins. It then attempts to call each of the plug-ins in the list until one of them successfully executes the desired operation. To appear in the list, a plug-in must meet the following criteria:

- It must be stored in the Extensions folder.
- It must be a shared library (file type `'shlb'`).
- It should have a unique creator type (registered with Apple).
- The name of the library's code fragment (specified in the `'cfrg'` resource) must be unique (among all currently loaded shared libraries) and begin with `'CSSP'`. For example, you might want to name the library by appending your creator type to `'CSSP'`.

Sample Scripts

The ColorSync SDK includes several sample scripts that demonstrate how to perform common operations. You can use the scripts as is, or borrow from them for your own custom scripts. For more information, see the detailed Read Me files that accompany the sample scripts.

Multiprocessor Support

Starting with version 2.5, ColorSync's default CMM can take advantage of multiple processors. Multiprocessor support is transparent to your code—the CMM invokes it automatically if the required conditions are met.

When ColorSync Uses Multiple Processors

The default CMM takes advantage of multiprocessor support only if the following conditions are satisfied:

1. The MPLibrary was successfully loaded at boot time.
2. The CMM successfully links against the MPLibrary at runtime.
3. The number of processors available is greater than one.
4. The number of rows in the image is greater than the number of processors.
5. The source and destination buffers have the same number of bytes per row or have different locations in memory.

Unless all of these conditions are met, matching will proceed without acceleration. Multiprocessor support is currently supplied only for the following component request codes:

- `kCMMMatchBitMap`
- `kCMMMatchPixMap`

As a result, the default CMM invokes multiprocessor support only in response to the general purpose `CWMatchPixMap` and `CWMatchBitmap` functions, or when those calls are invoked as a result of a call to the QuickDraw-specific matching routines, such as `NCMBeginMatching`.

Efficiency of ColorSync's Multiprocessor Support

Depending on the image and other factors, ColorSync's matching algorithms take advantage of multiple processors with up to 95% efficiency (your mileage may vary). If you have two processors, for example, ColorSync can complete a matching operation in as little 53% of the time required by one processor. Additional processors should provide proportional improvement.

QuickDraw GX and the ColorSync Manager

Unless your application uses QuickDraw GX to create and render images, your application must call ColorSync functions, such as `NCMBeginMatching` and `NCMDrawMatchedPicture`, to match colors between devices.

However, if your application uses QuickDraw GX and your application sets the view port attribute `gxEnableMatchPort`, the ColorSync Manager automatically matches colors when your application draws to the screen.

QuickDraw GX color profile objects contain ColorSync profiles, and each profile specifies the kind of matching to perform with it. For more information about QuickDraw GX color architecture, see the chapter Colors and Color-Related Objects in Inside Macintosh: QuickDraw GX Objects.

QuickDraw GX version 1.1.2 or earlier uses ColorSync 1.0. However, because the ColorSync Manager provides robust backward compatibility, including continued support of the ColorSync 1.0 API, you can use the ColorSync Manager with QuickDraw GX. For more information about the ColorSync Manager's backward compatibility, see Version and Compatibility Information (page 141).

Important: For information on changes to the printing and graphics architecture in the Mac OS that affect QuickDraw GX, see the technote <http://gemma.apple.com/technotes/gxchange.html>.

How the ColorSync Manager Uses Memory

The ColorSync Manager attempts to allocate the memory it requires from the following sources in this order:

- The current heap zone. If the current heap zone is set to the application heap, the ColorSync Manager will attempt to allocate the memory it requires from the application heap.
- The system heap. If the current heap zone is set to the system heap, the ColorSync Manager will try the system heap first and never attempt to allocate memory from the application heap.
- The Process Manager temporary heap. (If this final source does not satisfy the ColorSync's Manager's memory requirements, any attempt to load the ColorSync Manager will fail.)

By default, the current heap zone is set to the application heap. When the ColorSync Manager is used apart from QuickDraw GX, this scenario commonly prevails, making application heap memory available to the ColorSync Manager.

However, QuickDraw GX holds a covenant with applications committing not to allocate memory from the application heap. QuickDraw GX sets the current heap zone to the system heap. Consequently, when the ColorSync Manager is used by QuickDraw GX, the ColorSync Manager is prohibited from allocating memory it requires from the application heap and must allocate all the memory it requires from the system heap and the Process Manager temporary heap.

What Users Can Do With ColorSync-Supportive Applications

ColorSync allows your application or driver to maintain consistent color across devices and across platforms. You can also let users perform quick and inexpensive color proofing and see in advance which colors cannot be printed on their printers. This section provides an overview of these and other color management features you can provide. See Developing ColorSync-Supportive Applications (page 49) and Developing ColorSync-Supportive Device Drivers (page 109) for information on how to implement specific features.

Display Matching

When your application displays an image that contains one or more embedded profiles, it can use ColorSync to make sure the user experiences consistent color from one display to another. If a color cannot be reproduced exactly on a particular destination device, the ColorSync Manager can map the color to a similar color that is in the color gamut of the device.

Your application or driver can allow a user to embed or tag color-matching information and it should be able to use the ColorSync Manager to display a tagged picture. Most importantly, your application or driver must preserve picture comments in documents and allow the information to be passed on to the destination device.

Gamut Checking

Because not all colors can be rendered on all devices, you may want your application to warn users when a color they choose is out of gamut for the currently selected destination device. For example, you can use gamut checking to see if a given color is reproducible on a particular printer. If the color is not directly reproducible—that is, if it is out of gamut—you can alert the user to that fact. The ColorSync Manager provides the `CWCheckPixMap` and `CWCheckColors` functions for checking a color against a device's profile to see if it is in or out of gamut for the device. Your application can then display the results of this check to the user.

Soft Proofing

Using the destination device's profile, your application can enable users to preview on a monitor what a color image will look like on a particular device. Further, it enables remote proofing between client and prepress service. This simulation of a device's output can save the user considerable time and cost.

Device Link Profiles

Most users use the same device configuration for scanning, viewing, and printing over a period of time. Your application can allow users to create a device link profile. A device link profile is a means of storing in a single profile a series of linked profiles that correspond to a specific configuration in the order in which the devices in that configuration are normally used. A device link profile represents a one-way link or connection between devices. It does not represent any device model, nor can it be embedded into images.

Calibration

Your application can provide calibration services. A calibration application offers the option of calibrating a peripheral device based on a standard state or calibrating the device based on its current state.

If a peripheral device, such as a color printer, has drifted from its original state over time, a calibration application can make use of the characterization data contained in the corresponding profile to bring the color response back into range.

A user may want to improve the reproduction quality of a device without returning the device to a standard state. Your application can create a profile based on the current state of the device, then use the profile to characterize that device. This approach to calibration maintains the existing dynamic density range while improving the device's overall quality.

Note: You can also provide a monitor calibration plug-in, as described in Monitor Calibration and Profiles (page 41).

Developing ColorSync-Supportive Applications

This section describes how your application can use the ColorSync Manager to provide many color management services. For a complete list, see Developing Your ColorSync-Supportive Application (page 55).

Before you read this section, you should read Overview of Color and Color Management Systems (page 13) and Overview of ColorSync (page 27). These sections provide an overview of color theory and color management systems (CMSs), define key terms, and describe the ColorSync Manager.

If you are developing a device driver that supports ColorSync, you should read this section in addition to Developing ColorSync-Supportive Device Drivers (page 109).

If your application works with images created by other applications, you should at least read Providing Minimal ColorSync Support (page 56), which explains how to preserve profiles embedded in images.

While reading this section, refer to *ColorSync Manager Reference* for more information about the functions, constants, and data types used here.

ColorSync Version Information (page 141) describes the `Gestalt` information, shared library version numbers, CMM version numbers, and ColorSync header files you use with different versions of the ColorSync Manager. It also includes CPU and Mac OS system requirements.

The book Inside Macintosh: Imaging With QuickDraw describes how your application can use QuickDraw to create and display Macintosh graphics, and how to use the Printing Manager to print the images created with QuickDraw.

What's New (page 151) explains where to get information on the Color Picker Manager, which provides your application with a standard dialog box for soliciting a color choice from users.

You should read Important Note on Code Listings (page 11) before working with the code in this chapter.

About ColorSync Application Development

ColorSync provides your application with color-matching capabilities that users can employ without the need for a proprietary environment. ColorSync provides the first system-level implementation of an industry-standard color-matching system. Because ColorSync supports the profile format defined by the International Color Consortium (ICC), a color image a user creates can be color matched, rendered, and modified by another user running another application on another platform that supports the format. Conversely, your application can modify and color match images created by other applications that support ColorSync or a CMS that includes support for the ICC profile format. For information on profile format version numbers, see ColorSync and ICC Profile Format Version Numbers (page 30).

ColorSync Version Information (page 141) describes the `Gestalt` information, shared library version numbers, CMM version numbers, and ColorSync header files you use with different versions of the ColorSync Manager. It also describes CPU and Mac OS system requirements.

About the ColorSync Manager Programming Interface

The ColorSync Manager programming interface allows your application to handle tasks such as color matching, color conversion, profile management, profile searching and accessing, reading individual tagged elements within a profile, embedding profiles in documents, and modifying profiles.

The ColorSync API is summarized in Summary of the ColorSync Manager (page 107). You can find detailed information about individual functions, data types, and constants in *ColorSync Manager Reference*. The ColorSync Manager includes a number of interface files you may need for your development efforts. These files are described in ColorSync Header Files (page 143).

What Should a ColorSync-Supportive Application Do?

Your ColorSync-supportive application can provide a rich set of color-matching features. Your application can color match images, pixel maps, bitmaps, and even individual colors. In addition to color matching, you can handle such tasks as color conversion, color gamut checking, soft proofing of images, profile management, profile searching and accessing, reading individual tagged elements within a profile, embedding profiles and profile identifiers in documents, extracting embedded profiles and profile identifiers, and modifying profiles and profile identifiers.

Your application can provide an interface that offers pop-up menus or other user interface items allowing a user to choose which profile to associate with an image and how an image is rendered. It can show the user the colors of an image that are in or out of gamut for a particular device on which the image is to be produced and how ColorSync adjusts for colors that are out of gamut. This allows the user to preview differences that occur in the color-matching transition between gamuts and make corrections if necessary.

Most of the terms and operations mentioned in this section are defined in Overview of Color and Color Management Systems (page 13) and Overview of ColorSync (page 27).

At a Minimum

ColorSync allows your application to preserve high fidelity to the original colors of an image—whether the image was created using your application or another—by supporting the use of embedded profiles. Your application can take advantage of a profile embedded along with an image, matching the original colors of the device used to create the image to those of the destination display or printer. Even if your application doesn't support some of the more advanced features that ColorSync affords, such as soft proofing, you should color match images using the source profile, if one is identified and available.

At a minimum, your application should preserve images tagged with a profile by not stripping out picture comments used to embed profiles or by leaving profiles in documents that use other methods to include them.

It is important for your application to tag an image with the profile for the device used to create the image and to preserve existing tagging because a picture that is not tagged assumes use of a default profile as described in Setting Default Profiles (page 33). If the picture is moved to a different system that uses a different default profile, the picture will display differently. Providing Minimal ColorSync Support (page 56) explains how to preserve embedded profiles, and Embedding Profiles and Profile Identifiers (page 69) explains how to tag an image. Some of these features are described in greater detail in the rest of this material.

Storing and Handling Profiles

Profiles for use with the ColorSync Manager are stored in the ColorSync Profiles folder. The precise location of this folder can vary for different versions of ColorSync, as described in Setting Default Profiles (page 33). When you install ColorSync, the ColorSync Profiles folder contains a selection of display profiles for all Apple color monitors, as well as default profiles for standard color spaces and profiles for several Apple printers.

Starting with ColorSync 2.5, a user can select a default profile for certain color spaces from the ColorSync control panel, as described in Setting Default Profiles (page 33). Also starting with version 2.5, the Monitors & Sound control panel allows the user to select a separate profile for each monitor, as described in Monitor Calibration and Profiles (page 41).

Your application specifies the profiles for color matching when the application calls a ColorSync Manager function. For most functions, the ColorSync Manager uses one of the default profiles if your application doesn't specify a profile. Some functions require that you explicitly specify a profile by reference.

Device drivers for ColorSync-supportive input and output devices, such as scanners and printers, may install the profiles they use in the ColorSync Profiles folder, making them available to your application for color matching or gamut checking. If your application creates device link profiles, as described in Creating and Using Device Link Profiles (page 90), you should place those profiles in the ColorSync Profiles folder.

Your application can provide the interface to allow a user to choose a profile for a specific device. Using the ColorSync Manager functions described in *ColorSync Manager Reference*, your application can search the ColorSync Profiles folder and display information about available profiles.

See Developing Your ColorSync-Supportive Application (page 55) for a list of programming examples that demonstrate many of these features. As described in Providing Minimal ColorSync Support (page 56), your application should, at a minimum, leave profile information intact in the documents and pictures that it imports or copies into its own documents.

How the ColorSync Manager Selects a CMM

When a ColorSync function performs a color matching or color checking operation, it must determine which CMM to use. You typically pass source and destination profiles to a function, either directly or as part of a color world—an abstract private data structure you create by calling either the `NCWNewColorWorld`, the `CWConcatColorWorld`, or the `CWNewLinkProfile` function. When you call one of the latter two functions to create a color world, you use the `CMConcatProfileSet` data structure to specify a series of one or more profiles for the color world.

A profile header contains a `CMMType` field that specifies a CMM for that profile. For example, Signature of ColorSync's Default Color Manager describes a signature for the `CMMType` field that specifies ColorSync's default CMM. When you set up a `CMConcatProfileSet` data structure to specify a series of profiles, you set the structure's `keyIndex` field to specify the zero-based index of the profile within the array of profiles whose CMM (as indicated by its `CMMType` field) ColorSync should use. A CMM specified by this mechanism is called a key CMM.

As we have seen, an operation may use more than one profile and there are multiple factors that can affect the choice of a CMM. To deal with these factors, ColorSync uses the following algorithm to select a CMM:

1. Starting with version 2.5, a user can select a preferred CMM in the ColorSync control panel. If a user has chosen a preferred CMM, and if that CMM is available, ColorSync uses that CMM for all color checking and color matching operations the CMM can handle.

If the preferred CMM is not available or cannot handle an operation, ColorSync uses the default CMM, as described in step 4.

2. Prior to ColorSync 2.5, or if the user has not selected a preferred CMM with the ColorSync control panel, or has selected Automatic, and if the ColorSync function takes a color world reference and the user has initialized the color world with `CWConcatColorWorld` or `CWNewLinkProfile`, ColorSync uses the key CMM.

   If the key CMM is not available or cannot handle an operation, ColorSync uses the default CMM, as described in step 4

3. Prior to ColorSync 2.5, or if the user has not selected a preferred CMM with the ColorSync control panel, or has selected Automatic, and if the ColorSync function takes a color world reference and the user has initialized the color world with `NCWNewColorWorld` (and therefore without a `CMConcatProfileSet` structure), ColorSync uses an arbitrated CMM or CMMs—a CMM or CMMs selected from the source and destination profiles as described in Selecting a CMM by the Arbitration Algorithm (page 52).

   If an arbitrated CMM is not available or cannot handle an operation, ColorSync uses the default CMM, as described in step 4.

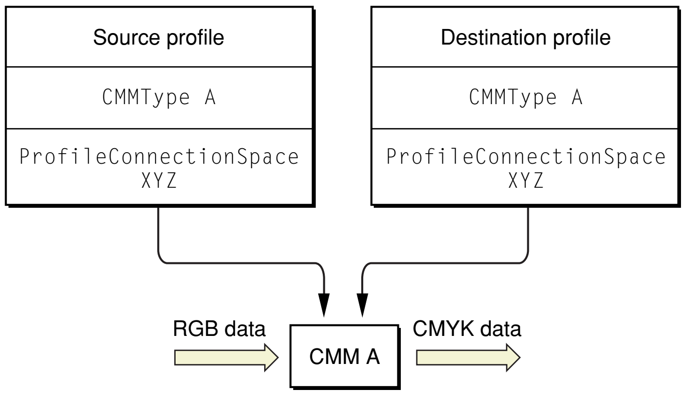
4. If a CMM is not specified by one of the previous three steps, or if a specified CMM is not available or cannot handle an operation, ColorSync uses the default CMM—the robust CMM that is installed as part of the ColorSync extension. The default CMM supports all the required and optional functions defined by the ColorSync Manager, and is therefore a suitable CMM of last resort. The signature for the default CMM is specified by the constant kDefaultCMMSignature.

Selecting a CMM by the Arbitration Algorithm

This section describes the arbitration algorithm, introduced in How the ColorSync Manager Selects a CMM (page 51), ColorSync uses to select one or more arbitrated CMMs for a color matching or color checking operation:

- If the source and destination profiles specify the same CMM and that CMM component is available and able to perform the matching, then the specified CMM maps the colors directly from the color space of the source profile to the color space of the destination profile. This is the simplest scenario, and Figure 4-1 (page 52) illustrates it.

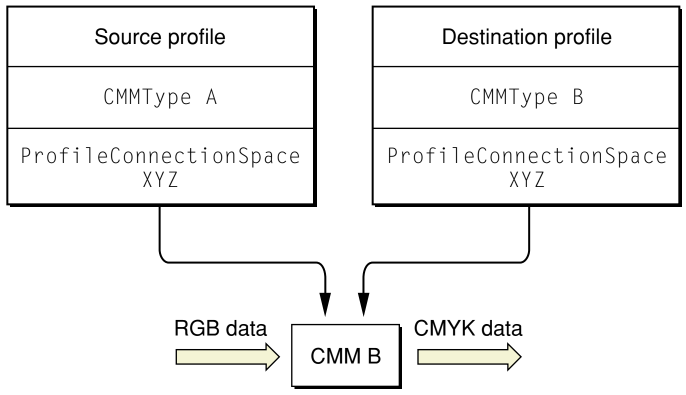
Figure 4-1 Colormatching when the source and destination profiles specify the same CMM



- If the source and destination profiles specify different CMMs, then the ColorSync Manager follows these steps to choose the CMM:

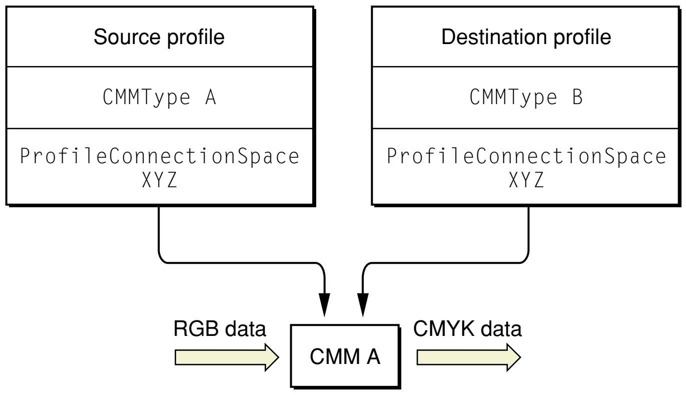
  1. If the CMM specified by the destination profile is available, is able to perform the color matching using the two profiles, and is not the default CMM, then the ColorSync Manager uses this CMM. Figure 4-2 (page 53) shows this scenario.

Figure 4-2 Color matching using the destination profile's CMM



  1. If the destination profile's specified CMM is unavailable or unable to perform the color-matching request using the two profiles, then the ColorSync Manager looks for the CMM specified by the source profile. If the CMM specified by the source profile is available, is able to perform the color matching using the two profiles, and is not the default CMM, the ColorSync Manager uses this CMM. Figure 4-3 (page 53) shows this scenario.

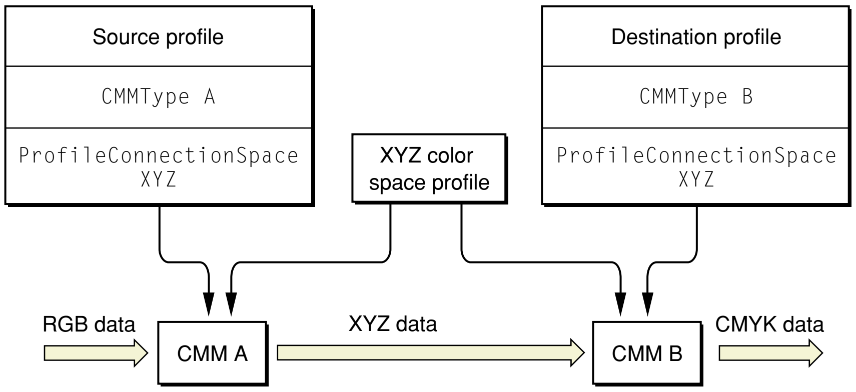
Figure 4-3 Color matching using the source profile's CMM



  1. If both the source-specified CMM and the destination-specified CMM are available, but neither is able to perform the match alone, the ColorSync Manager uses the source profile's CMM to convert the colors of the source image from the source profile's color space to an interchange color space using the XYZ color space profile as the destination profile. Next, the ColorSync Manager uses the CMM specified by the destination profile to convert the colors now specified in the interchange

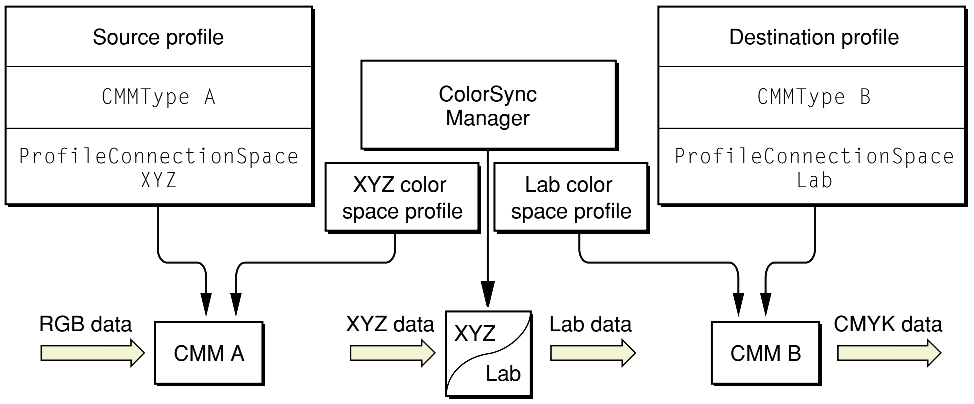
color space to colors expressed in the color space of the destination profile using the XYZ color space profile as the source profile. The color conversion and matching work this way if both profiles specify the same interchange color space. Figure 4-4 (page 54) shows this scenario.

Figure 4-4 Color matching through an XYZ interchange space using both CMMs



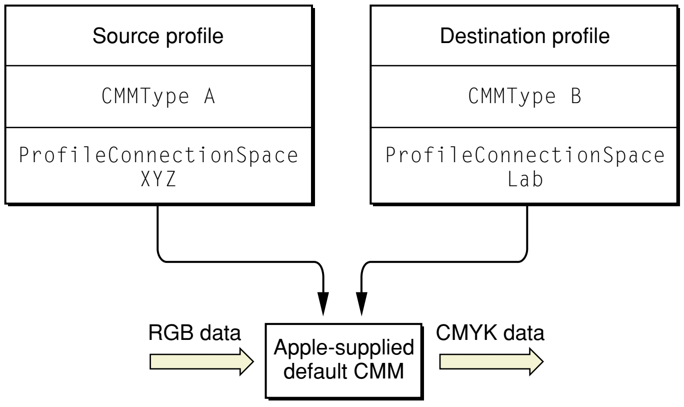
1. If both the source-specified CMM and the destination-specified CMM are available, but neither is able to perform the match alone and both profiles specify different interchange color spaces, the ColorSync Manager uses the source profile's CMM to convert the colors of the source image from the source profile's color space to its interchange color space using the appropriate color space profile as the destination profile. The example shown in Figure 4-5 (page 54) uses the XYZ color space profile as the destination profile. Then the ColorSync Manager inserts a part into the process, itself converting colors from the source profile's interchange color space to the destination profile's interchange color space. Next, the ColorSync Manager uses the CMM specified by the destination profile to convert the colors now specified in the destination profile's interchange color space to colors expressed in the destination profile's color space using the appropriate color space profile as the source profile. The example shown in Figure 4-5 (page 54) uses the Lab color space profile as the source profile.

Figure 4-5 Matching using both CMMs and two interchange color spaces



- If neither the source nor the destination profile's specified CMM is available or able to perform the color conversion and matching, then the ColorSync Manager uses the default CMM, which will always attempt to perform the match. Figure 4-6 (page 55) shows this scenario.

Figure 4-6 Color matching using the default CMM


Source profile
CMMType A
ProfileConnectionSpace XYZ
Destination profile
CMMType B
ProfileConnectionSpace Lab
RGB data
Apple-supplied default CMM
CMYK data

Developing Your ColorSync-Supportive Application

This section describes some of the tasks your application can perform to implement color-matching and color-checking features with the ColorSync Manager.

This section provides code samples for:

- Determining If the ColorSync Manager Is Available (page 56); revised for ColorSync 2.5
- Providing Minimal ColorSync Support (page 56)
- Opening a Profile and Obtaining a Reference to It (page 58); revised for ColorSync 2.5
- Identifying the Current System Profile (page 60)
- Poor Man's Exception Handling (page 60); new for ColorSync 2.5
- Getting the Profile for the Main Display (page 61); revised for ColorSync 2.5
- Matching to Displays Using QuickDraw-Specific Operations (page 62)
- Creating a Color World to Use With the General Purpose Functions (page 64)
- Matching Colors Using the General Purpose Functions (page 66)
- Embedding Profiles and Profile Identifiers (page 69)
- Extracting Profiles Embedded in Pictures (page 73)
- Performing Optimized Profile Searching (page 81); new for ColorSync 2.5
- Searching for Specific Profiles Prior to ColorSync 2.5 (page 86)
- Searching for a Profile That Matches a Profile Identifier (page 87)
- Checking Colors Against a Destination Device's Gamut (page 90)

Determining If the ColorSync Manager Is Available

To determine whether version 2.5 of the ColorSync Manager is available on a 68K-based or a PowerPC-based Macintosh system, you use the `Gestalt` function with the `gestaltColorMatchingVersion` selector. The function shown in Listing 4-1 (page 56) returns a Boolean value of `true` if version 2.5 or later of the ColorSync Manager is installed and `false` if not.

Listing 4-1 Determining if ColorSync 2.5 is available

```
Boolean ColorSync25Available (void)
{
    Boolean haveColorSync25 = false;
    long    version;

    if (Gestalt(gestaltColorMatchingVersion, &version) == noErr)
    {
        if (version >= gestaltColorSync25)
        {
            haveColorSync25 = true;
        }
    }
    return haveColorSync25;
}
```

If your application does not depend on features added for version 2.5 of the ColorSync Manager, use the ColorSync Gestalt selector for the ColorSync version you require. For example, you might substitute gestaltColorSync20 for gestaltColorSync25 in the previous function (and rename the function appropriately). To identify other versions of ColorSync, use any of the ColorSync Gestalt selector constants described in *ColorSync Manager Reference*. For related version information, see ColorSync Version Information (page 141).

Providing Minimal ColorSync Support

ColorSync supports the profile format defined by the International Color Consortium (ICC). The ICC format provides a single cross-platform standard for translating color data across devices. The ICC's common profile format allows one user to electronically transfer a document containing a color image to another user with the assurance that the original image will be rendered faithfully according to the source profile for the image.

To ensure this, the application or driver used to create the image stores the profile for the source device in the document containing the color image. The application can do this automatically or allow the user to tag the image. If the source profile is embedded within the document, a user can move the document from one system to another without concern for whether the profile used to create the image is available.

To support ColorSync, your application should, at a minimum, leave profile information intact in the documents and pictures it imports or copies. That is, your application should not strip out profile information from documents or pictures created with other applications. Even if your application does not use the profile information, users may be able to take advantage of it when using the documents or pictures with other applications.

For example, profiles and profile identifiers may be embedded in pictures that a user pastes into documents created by your application. A profile identifier is an abbreviated data structure that identifies, and possibly modifies, a profile in memory or on disk. For more information on profile identifiers, see Searching for a Profile That Matches a Profile Identifier (page 87). Profiles and profile identifiers can be embedded in formats such as PICT or TIFF files. For files of type `'PICT'`, the ColorSync Manager defines the following picture comments for embedding profiles and profile identifiers, and for performing color matching:

```
/* PicComment IDs */
enum {
    cmBeginProfile          = 220,  /* begin ColorSync 1.0 profile */
    cmEndProfile            = 221,  /* end a ColorSync 2.x or 1.0
                                        profile */
    cmEnableMatching        = 222,  /* begin color matching for either
                                        ColorSync 2.x or 1.0 */
    cmDisableMatching       = 223,  /* end color matching for either
                                        ColorSync 2.x or 1.0 */
    cmComment               = 224   /* embedded ColorSync 2.x profile
                                        information */
};
```

The picture comment `kind` value of cmComment is defined for embedded ColorSync Manager version 2.x profiles and profile identifiers. This picture comment is followed by a 4-byte selector that describes the type of data in the picture comment.

```
/* PicComment selectors for cmComment */
enum {
    cmBeginProfileSel       = 0,    /* begining of a ColorSync 2.x
                                        profile; profile data to
                                        follow */
    cmContinueProfileSel    = 1,    /* continuation of a ColorSync
                                        2.x profile; profile data to
                                        follow */
    cmEndProfileSel         = 2     /* end of ColorSync 2.x profile
                                        data; no profile data follows */
    cmProfileIdentifierSel  = 3     /* profile identifier information
                                        follows; the matching profile
                                        may be stored in the image or
                                        on disk */
};
```

Your application should leave these comments and the embedded profile information they define intact. Similarly, if your application imports or converts file types defined by other applications, your application should maintain the profile information embedded in those files, too.

Your application can also embed picture comments and profiles in documents and pictures it creates or modifies. For information describing how to do this, see Embedding Profiles and Profile Identifiers (page 69). Inside Macintosh: Imaging With QuickDraw describes picture comments in detail.

Obtaining Profile References

Most of the ColorSync Manager functions require that your application identify the profile or profiles to use in carrying out the work of the function. For example, when your application calls functions to perform color matching or color gamut checking, you must identify the profiles to use for the session. For functions that use QuickDraw, you specify a source profile and a destination profile. For general purpose functions, you specify a color world containing source and destination profiles or a set of concatenated profiles. You can also create a device link profile, which is described in Creating and Using Device Link Profiles (page 90), but to do so your application must first obtain references to all the profiles that will comprise the device link profile.

The ColorSync Manager provides for multiple concurrent accesses to a single profile through use of a private data structure called a profile reference. A profile reference is a unique reference to a profile; it is the means by which your application identifies a profile and gains access to the contents of that profile. Many applications can use the same profile at the same time, each with its own reference to the profile. However, an application can only change a profile if it has the only reference to the profile.

Opening a Profile and Obtaining a Reference to It

To open a profile and obtain a reference to it, you call the function `CMOpenProfile`. You can also obtain a profile reference from the `CMCopyProfile`, `CWNewLinkProfile`, and `CMNewProfile` functions. To identify a profile that is file based, memory based, or accessed through a procedure, you must give its location.

The ColorSync Manager defines the `CMProfileLocation` data type to specify a profile's location:

```
struct CMProfileLocation {
    short        locType;   /* specifies the location type */
    CMProfLoc    u;         /* structure for specified type */
};
```

The `CMProfileLocation` structure contains the u field of type `CMProfLoc`. The `CMProfLoc` type is a union that can provide access to any of the structures `CMFileLocation`, `CMHandleLocation`, `CMPtrLocation`, or `CMProcedureLocation`.

The data you specify in the u field indicates the actual location of the profile. In most cases, a ColorSync profile is stored in a disk file and you use the union for a file specification. However, a profile can also be located in memory, or in an arbitrary location (such as a resource) that is accessed through a procedure provided by your application. For more information on profile access, see Accessing a Resource-Based Profile With a Procedure (page 94). In addition, you can specify that a profile is temporary, meaning that it will not persist in memory after your application uses it for a color session.

To identify the data type in the `u` field of the `CMProfileLocation` structure, you assign to the `CMProfileLocation.locType` field one of the constants or numeric equivalents defined by the following enumeration:

```
enum {
    cmNoProfileBase         = 0,    /* the profile is temporary */
    cmFileBasedProfile      = 1,    /* file-based profile */
    cmHandleBasedProfile    = 2,    /* handle-based profile */
    cmPtrBasedProfile       = 3     /* pointer-based profile */
    cmProcedureBasedProfile = 4     /* procedure-based profile */
};
```

For example, for a file-based profile, the u field would hold a file specification and the locType field would hold the constant cmFileBasedProfile. Your application passes a CMProfileLocation structure when it calls the CMOpenProfile function and the function returns a reference to the specified profile.

Note: If you already have a profile reference for a profile, you can call the NCMGetProfileLocation function (available starting in ColorSync 2.5) or the CMGetProfileLocation function (for previous versions of ColorSync) to obtain the location for the profile.

Listing 4-2 (page 59) shows an application-defined function, MyOpenProfileFSSpec, that assigns the file specification for a profile file to the profLoc union and identifies the location type as file-based. It then calls the CMOpenProfile function, passing to it the profile's file specification and receiving in return a reference to the profile.

Listing 4-2 Opening a reference to a file-based profile

```
CMError MyOpenProfileFSSpec (FSSpec spec, CMProfileRef *prof)
{
    CMError                 theErr;
    CMProfileLocation       profLoc;

    profLoc.u.fileLoc.spec = spec;
    profLoc.locType = cmFileBasedProfile;

    theErr = CMOpenProfile(prof, &profLoc);

    return theErr;
}
```

Reference Counts for Profile References

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

When your application passes a copy of a profile reference to an independent task, whether synchronous or asynchronous, the task should call CMCloneProfileRef to increment the reference count. Both the called task and the caller should call CMCloseProfile when finished with the profile reference. This ensures that the tasks can finish independently of each other.

Important: 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 and only once for the profile.

Important: 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 a task may attempt to use a profile reference that is no longer valid.

Poor Man's Exception Handling

Listing 4-3 (page 60) shows a macro definition that is used in several subsequent code listings. In this macro, if `assertion` evaluates to `false`, execution continues at the location `exception`. Otherwise, execution continues at the next statement following the macro.

Listing 4-3 Poor man's exception handling macro

```
// Equivalent to  if ((assertion) == false) goto exception;
#define require(assertion, exception)    \
    do {                                 \
        if (assertion) ;                 \
        else { goto exception; }         \
    } while (false)
```

You can find examples of how to use this macro in Listing 4-4 (page 61), Listing 4-5 (page 61) and others. While this style of poor man's exception handling may not appeal to all developers, it does offer these advantages:

- It improves readability because it avoids pushing code off the page with multiple nested if then else clauses and by limiting the number of `return` statements in the code.
- You can enhance the macro to provide debug messages that supply useful runtime information about the error or where it occurred.

Identifying the Current System Profile

For the functions `NCMBeginMatching`, `NCMUseProfileComment`, and `NCWNewColorWorld`, your application can specify `NULL` to signify the system profile. For all other functions—for example, the `CMGetProfileElement` function, the `CMValidateProfile` function, and the `CMCopyProfile` function—for which you want to specify the system profile, you must give an explicit reference to the profile. You can use the `CMGetSystemProfile` function to obtain a reference to the system profile.

Important: Starting with ColorSync version 2.5, the system profile is used primarily for backward compatibility, as described in Setting Default Profiles (page 33). As a result, you should not use the system profile as a source or destination profile if you can determine a specific profile to use instead. For example, you may want to call `CMGetDefaultProfileBySpace` to get the default profile for a specific color space or `CMGetProfileByAVID` to get a profile for a specific display.

Each profile, including the profile configured as the system profile, has a name associated with it. If your application needs to display the name of the system profile to the user, it can call CMGetSystemProfile, as shown in Figure 4-4 (page 54), to get the system profile, then call the `CMGetScriptProfileDescription` function to get the profile name and script code.

Listing 4-4 Identifying the current system profile

```
CMError MyPrintSystemProfileName (void)
{
    CMError         theErr;
    CMProfileRef    sysProf;
    Str255          profName;
    ScriptCode      profScript;
    theErr = CMGetSystemProfile(&sysProf);
    require(theErr == noErr, cleanup);

    theErr = CMGetScriptProfileDescription(sysProf, profName,
                                                &profScript);
    require(theErr == noErr, cleanup);
    // … call Script Mgr to get correct font for script …

    DrawString(profname);

// Do any necessary cleanup. In this case, just return.
cleanup:
    return theErr;
}
```

Getting the Profile for the Main Display

Starting with ColorSync version 2.5, a user can select a separate profile for each display, as described in Setting a Profile for Each Monitor (page 42). In your code, you can determine the profile for any display for which you know the AVID by calling the function `CMGetProfileByAVID`, which is also new in version 2.5. You can get more information about AVID values from the Display Manager SDK.

Listing 4-5 (page 61) shows how to get the profile for the main display (the one that contains the menu bar).

Listing 4-5 Getting the profile for the main display

```
CMError GetProfileForMainDisplay (CMProfileRef *prof)
{
    CMError     theErr;
    AVIDType    theAVID;
    GDHandle    theDevice;

    // Get the main GDevice.
    theDevice = GetMainDevice();

    // Get the AVID for that device.
    theErr = DMGetDisplayIDByGDevice(theDevice, &theAVID, true);
    require(theErr == noErr, cleanup);

    // Get the profile for that AVID.
    theErr = GetProfileByAVID(theAVID, prof);
    require(theErr == noErr, cleanup);

// Do any necessary cleanup. In this case, just return.
cleanup:
    return theErr;
}
```

This code first gets a graphic device handle for the main display, then calls the Display Manager routine `DMGetDisplayIDByGDevice` to get an AVID for the device. It then passes the AVID to the ColorSync Manager routine `CMGetProfileByAVID` to get a profile reference to the profile for the display.

Matching to Displays Using QuickDraw-Specific Operations

To provide images and pictures showing consistent colors across displays, your application can use ColorSync to match the colors in a user's pictures and documents with the colors available on the user's current display. If a color cannot be reproduced on the system's current display, ColorSync maps the color to the color gamut of the display according to the specifications defined by the profiles. When Color Matching Occurs (page 38) describes both QuickDraw-specific and general purpose ColorSync functions for color matching.

The ColorSync Manager provides two QuickDraw-specific functions that your application can call to draw a color picture to the current display. The function `NCMDrawMatchedPicture` matches the picture's colors to the display's gamut defined by the specified display profile. It uses the system profile as the initial source profile but switches to any embedded profiles as they are encountered. The function `NCMBeginMatching` uses the source and destination profiles you specify to match the colors of the source image to the colors of the device for which it is destined.

The current display device's profile is typically configured as the system profile. A user can do this with the ColorSync control panel. However, starting with ColorSync 2.5, a user can use the Monitors & Sound control panel to set a separate profile for each display, as described in Setting a Profile for Each Monitor (page 42). When a user sets a profile for a display, ColorSync makes that profile the current default system profile.

Because the ColorSync Manager assumes the system profile is that of the current display, you can pass a value of `NULL` to the QuickDraw-specific functions instead of supplying an explicit profile reference. Passing `NULL` for a profile reference directs the ColorSync Manager to use the system profile. Note however, that starting with ColorSync 2.5, if you know the primary display for the image, and you know the AVID for that display, you can call `CMGetProfileByAVID` to get the profile for the specific display. For example, Listing 4-5 (page 61) shows how to get the profile for the main display (the one with the menu bar).

The following sections describe how to use ColorSync's QuickDraw-specific matching functions, which automatically perform color matching in a manner acceptable to most applications. However, if your application needs a finer level of control over color matching than is supplied by the QuickDraw-specific functions, you can use the general purpose functions described inMatching Colors Using the General Purpose Functions (page 66) to match the colors of a bitmap, a pixel map, or a list of colors.

Matching Colors in a Picture Containing an Embedded Information

If a user copies a picture that includes a profile or profile identifier into one of your application's documents, your application can use the ColorSync Manager's QuickDraw-specific function `NCMDrawMatchedPicture` to match the colors in that picture to the display on which you draw it.

As the picture is drawn, the `NCMDrawMatchedPicture` function automatically matches all colors to the color gamut of the display device, using the destination profile passed in the `dst` parameter. To use this function, you need to supply only the profile for the destination display device. The function acknowledges color-matching picture comments embedded in the picture and uses embedded profiles and profile identifiers. The source profile for the device on which the image was created should be embedded in the QuickDraw picture whose handle you pass to the function; the `NCMDrawMatchedPicture` function uses the embedded source profile, if it exists. If the source profile is not embedded, the function uses the current system profile as the source profile.

A picture may have more than one profile embedded, and may embed profile identifiers that refer to, and possibly modify, embedded profiles or profiles on disk. If the profiles and profile identifiers are embedded correctly, the `NCMDrawMatchedPicture` function will use them successively, as they are encountered.

By specifying `NULL` as the destination profile when you use this function, you are assured that the system profile—typically set to the profile for the main screen—is used as the destination profile. Alternatively, your application can call the `CMGetSystemProfile` function to obtain a reference to the profile and specify the system profile explicitly. Or, starting in ColorSync version 2.5, if you know the AVID for the display on which drawing takes place, you can call `CMGetProfileByAVID` to get the profile for the display.

Listing 4-6 (page 63) shows sample code that uses the QuickDraw-specific function `NCMDrawMatchedPicture` to perform color matching to a display. The code gets a profile for the destination display using an AVID if it is available; otherwise, it passes `NULL` to the `NCMDrawMatchedPicture` function to specify the system profile.

Listing 4-6 Matching a picture to a display

```
// Matching a picture to a display
CMError MyDrawPictureToADisplay (PicHandle thePict, AVIDType theAVID, Rect
*destRect)
{
    CMError         theErr;
    CMProfileRef    destProf;

    // Init for error handling.
    theErr = noErr;
    destProf = NULL;

    // If caller supplied an AVID and CS 2.5 is running...
    if (theAVID && ColorSync25Available() )
    {
        theErr = GetProfileByAVID(theAVID, &destProf);
        require(theErr == noErr, cleanup);
    }
    else
    {
        // Use the System profile as the destination.
        destProf = NULL;
    }
    // Draw the picture, with color matching.
    NCMDrawMatchedPicture(thePict, destProf, destRect);
    theErr = QDError();
    require(theErr == noErr, cleanup);

// Do any necessary cleanup. If necessary, close the profile.
cleanup:

    if (destProf)
        CMCloseProfile(destProf);

    return theErr;
}
```

More on Embedded Information

For embedded profiles (and profile identifiers) to operate correctly, the currently effective profile must be terminated by a picture comment of `kindcmEndProfile` after drawing operations using that profile are performed. If a picture comment was not specified to end the profile, the profile will remain in effect until the next embedded profile is encountered with a picture comment of `kind cmBeginProfile`. However, use of the next profile might not be the intended action. It is good practice to always pair use of the `cmBeginProfile` and `cmEndProfile` picture comments. When the ColorSync Manager encounters an `cmEndProfile` picture comment, it restores use of the system profile for matching until it encounters another `cmBeginProfile` picture comment.

Note: Profile identifiers are also stored with picture comments. For more information on profile identifiers, see Embedding Profiles and Profile Identifiers (page 69) and Searching for a Profile That Matches a Profile Identifier (page 87).

If your application allows a user to modify and save an image that you color matched using the function `NCMUseProfileComment`, your application should either embed the destination profile in the picture file or convert and match the colors of the modified image to the colors of the source profile. By doing this your application ensures the integrity of the image during future operations and display. The method you choose is specific to your application.

Matching Colors as a User Draws a Picture

To use Color QuickDraw functions to draw a document with colors matched to a display, your application can simply use the `NCMBeginMatching` function before calling Color QuickDraw functions, then conclude its drawing with the `CMEndMatching` function. For example, you might want to do this to customize settings in the profile that affect the matching operation. For more information on Color QuickDraw drawing functions, see Inside Macintosh: Imaging With QuickDraw.

To use the `NCMBeginMatching` function, you must specify both the source and destination profiles. The `NCMBeginMatching` function returns a reference to the color-matching session in its myRef parameter. You then pass the reference to the `CMEndMatching` function to terminate color matching. Code for performing this operation is not shown here.

Creating a Color World to Use With the General Purpose Functions

A color world is a reference to a private ColorSync structure that represents a unique color-matching session. Although profiles can be large, a color world is a compact representation of the mapping needed to match between profiles. Conceptually, you can think of a color world as a sort of matrix multiplication of two or more profiles that distills all the information contained in the profiles into a fast, multidimensional lookup table.

For the ColorSync Manager general purpose functions, a color world characterizes how the color-matching session will occur based on information contained in the profiles that you supply when your application sets up the color world. When Color Matching Occurs (page 38) describes both general purpose and QuickDraw-specific ColorSync functions for color matching. Your application can define a color world for color transformations between a source profile and a destination profile, or it can define a color world for color transformations between a series of concatenated profiles.

For the general purpose ColorSync Manager functions, a color world is the equivalent of the ColorSync Manager QuickDraw-based functions' source and destination profiles. From your application's perspective, the difference in specifying profiles for the general purpose functions is that instead of calling a function and passing it references to the profiles for the session, first you must create a color world using those profile references and pass the color world to the function. This general purpose interface provides better performance during color-matching.

Your application calls the `NCWNewColorWorld` function to set up a simple color world for color transformations involving two profiles—a source profile and a destination profile—and the function returns a reference to the color world it creates. Setting up a color world for color processing involving a series of concatenated profiles or a single device link profile, which contains a series of profiles, is slightly more complex. Here are the steps you take:

1. Obtain references to the profiles to use for the concatenated color world.

   For information describing how to obtain references to the profiles for the color world, see Obtaining Profile References (page 58).

2. Set up an array containing references to the profiles comprising the set.

   Before your application calls the function `CWConcatColorWorld` to create the color world, you must establish the profile set. The ColorSync Manager defines the following data structure of type `CMConcatProfileSet` that you use to specify the profile set:

   ```
   struct CMConcatProfileSet { unsigned shortkeyIndex; unsigned shortcount;
   CMProfileRefprofileSet[1]; };
   ```

   Your application also uses the `CMConcatProfileSet` data structure to define a profile set for a device link profile. See Creating and Using Device Link Profiles (page 90) for more information.

   Your application creates an array that contains references to the profiles for the color world, specifying these references in processing order. You specify the one-based number of profile references in the array by setting the value of the `CMConcatProfileSet.count` field. You assign the profile array to the `CMConcatProfileSet.profileSet` field.

   The ColorSync Manager defines rules governing the types of profiles you can specify in a profile array. These rules differ depending on whether you are creating a profile set to create a device link profile or to create a concatenated color world. For a list of the rules defining the types of profiles you can use for these purposes, see `CWNewLinkProfile` and `CWConcatColorWorld`.

3. Identify the CMM for color processing.

   Each of the profiles whose references you give identifies a CMM for color processing involving that profile. To perform color transformation using a series of profiles, the ColorSync Manager uses only one CMM. You use the `CMConcatProfileSet.keyIndex` field to identify the index into the array corresponding to the profile whose specified CMM is to be used. The array is zero based, so you must specify the `CMConcatProfileSet.keyIndex` value as a number in the range of 0 to `count` – 1, where `count` is the number of elements in the array.

   Important: See How the ColorSync Manager Selects a CMM (page 51) for a complete description of the ColorSync algorithm for selecting a CMM.

4. Call the CWConcatColorWorld function to set up the color world.

You pass the `CWConcatColorWorld` function a parameter of type `CMConcatProfileSet` to specify the profile array, and the function returns a color world reference. To perform color matching or gamut checking using the profiles comprising a color world, you call the general purpose function passing it the reference to the color world.

Using a device link profile for the general purpose functions entails additional steps, described in Creating and Using Device Link Profiles (page 90).

Matching Colors Using the General Purpose Functions

When Color Matching Occurs (page 38) describes both general purpose and QuickDraw-specific ColorSync functions for color matching. Using the general purpose functions `CWMatchPixMap` or `CWMatchBitmap`, your application can match the colors of a pixel image or a bitmap image to the display's color gamut without relying on QuickDraw.

Color matching occurs relatively quickly, but for a session involving a large pixel image or bitmap image, the color-matching process may take some time. To keep the user informed, you can provide a progress-reporting function. For example, your function can display an indicator, such as a progress bar, to depict how much of the matching has been done and how much remains. Your function can also allow the user to interrupt the color-matching process.

When your application calls either the `CWMatchPixMap` function or the `CWMatchBitmap` function, you can pass the function a pointer to your callback progress-reporting function and a reference constant containing data, such as the progress bar dialog box's window reference. When the CMM used to match the colors calls your progress-reporting function, it passes the reference constant to it. If you provide a progress-reporting function, here is how you should declare the function, assuming you name it `MyCMBitmapCallBackProc`:

```
pascal Boolean MyCMBitmapCallBackProc (long progress, void *refCon);
```

For a complete description of the progress-reporting function declaration, see `MyCMBitmapCallBackProc`.

To use the `CWMatchPixMap` and `CWMatchBitmap` functions, your application must first set up a color world that specifies the profiles involved in the color-matching session as described in Creating a Color World to Use With the General Purpose Functions (page 64). The color world establishes how matching will take place between the profiles. Listing 4-7 (page 67) shows how to match the colors of a bitmap using the general purpose functions that take a color world.

The ColorSync Manager uses the `PixMap` data type defined by Color QuickDraw. The ColorSync Manager defines and uses the `cmBitmap` data type, based on the classic QuickDraw `Bitmap` data type.

Matching the Colors of a Pixel Map to the Display's Color Gamut

Your application can call the function `CWMatchPixMap` to match the colors of a pixel image to the display's color gamut. To use `CWMatchPixMap`, you first create a color world, as described in Creating a Color World to Use With the General Purpose Functions (page 64). The color world is based on the source profile for the device used to create the pixel image and the destination profile for the display on which the image is shown.

To match the colors of a pixel image to the display's color gamut, the source profile for the color world must specify a data color space of RGB as its `dataColorSpace` element value to correspond to the pixel map data type, which is implicitly RGB. If the source profile you specify for the color world is the original source profile used to create the pixel image, most likely these values match. However, if you want to verify that the source profile's `dataColorSpace` element specifies RGB, you can use the `CMGetProfileHeader` function to obtain the profile header. The profile header contains the `dataColorSpace` element field. For a pixel image, the display profile's `dataColorSpace` element must also be set to RGB; this is the color space commonly used for displays.

If the source profile is embedded in the document containing the pixel map, your application can extract the profile and open a reference to it before you create the color world. For information on how to extract an embedded profile, see Extracting Profiles Embedded in Pictures (page 73). If the source profile is installed in the ColorSync Profiles folder, your application can display a list of profiles to the user to allow the user to select the appropriate one.

Matching the Colors of a Bitmap Image to the Display's Color Gamut

Matching the colors of a bitmap image to the current system's display is similar to the process of matching a pixel map's colors, except that the data type of a bitmap image is explicitly stated in the `space` field of the bitmap. You can specify a bitmap image using any of the following data types: `cmGraySpace`, `cmGrayASpace`, `cmRGB16Space`, cmRGB24Space, `cmRGB32Space`, `cmARGB32Space`, `cmRGB48Space`, `cmCMYK32Space`, `cmCMYK64Space`, `cmHSV32Space`, `cmHLS32Space`, `cmYXY32Space`, `cmXYZ32Space`, `cmLUV32Space`, `cmLAB24Space`, `cmLab32Space`, `cmLAB48Space`, `cmNamedIndexed32Space`, `cmMCFive8Space`, `cmMCSix8Space`, `cmMCSeven8Space`, or `cmMCEight8Space`. The data type of the source bitmap image must correspond to the data color space specified by the color world's source profile.

When you call the `CWMatchBitmap` function, you can pass it a pointer to a bitmap to hold the resulting image. In this case, you must allocate the pixel buffer pointed to by the `image` field of the `CMBitmap` structure. Because the `CWMatchBitmap` function allows you to specify a separate bitmap to hold the resulting color-matched image, you must ensure that the data type you specify in the `space` field of the resulting bitmap matches the destination's color data space. On input, the color space of the source profile must match the color space of the bitmap. If you specify `NULL` for the destination bitmap, on successful output, ColorSync will change the `space` field of the source bitmap to reflect the bitmap space to which the source image was mapped.

Rather than create a bitmap for the color-matched image, you can match the bitmap in place. To do so, you specify `NULL` instead of passing a pointer to a resulting bitmap.

The code in Listing 4-7 (page 67) shows how to set up a bitmap for the resulting color-matched image before calling the `CWMatchBitmap` function to perform the color matching. The `MyMatchImage` function calls the `MyGetImageProfile` function (not shown) to obtain an embedded profile from the image. If none is found, it calls the `MyGetImageSpace` function (also not shown) to determine the color space for the profile, then calls the ColorSync routine `CMGetDefaultProfileBySpace` to obtain the default profile for that space.

The `MyMatchImage` function then calls `GetProfileForMainDisplay`, shown in Listing 4-5 (page 61), to get the destination profile. It uses the source and destination profiles to set up a color world by calling `NCWNewColorWorld`, then uses the resulting color world when it calls `CWMatchBitmap` to match the colors to the display.

Listing 4-7 Matching the colors of a bitmap using a color world

```
void MyMatchImage (FSSpec theImage)
```

```
{
    CMError         theErr;
    CMProfileRef    sourceProf;
    CMProfileRef    destProf;
    CMWorldRef      cw;
    CMBitmap        bitmap;
    OSType          theSpace;

    /* Init for error handling. If any error during process,
        jump to cleanup area and quit trying. */
    theErr = noErr;
    sourceProf = nil;
    destProf = nil;
    cw = nil;
    bitmap.image = nil;

    // Determine source profile.
    // 1st - try to find an embedded profile
    theErr = MyGetImageProfile(theImage, &sourceProf);
    if (theErr == noErr)
    {
        // 2nd - use default profile for the image space
        theErr = MyGetImageSpace(theSpace, &sourceProf);
        require(theErr == noErr, cleanup);

        theErr = CMGetDefaultProfileBySpace(theSpace, &sourceProf);
        require(theErr == noErr, cleanup);
    }
    require(theErr == noErr, cleanup);

    // Determine dest profile.
    theErr = GetProfileForMainDisplay(&destProf);
    require(theErr == noErr, cleanup);

    // Set up a color world.
    theErr = NCWNewColorWorld(&cw, sourceProf, destProf);
    require(theErr == noErr, cleanup);

    // close profiles after setting up color world.
    if (sourceProf)
        CMCloseProfile(sourceProf);
    if (destProf)
        CMCloseProfile(destProf);
    sourceProf = destProf = nil;

    // Read the image into the CMBitmap structure
    theErr = MyGetImageBitmap(theImage, &bitmap);
    require(theErr == noErr, cleanup);

    // Match bitmap in place.
    theErr = CWMatchBitmap(cw, &bitmap, nil, nil, nil);
    require(theErr == noErr, cleanup);

    // Render results here ... (code not shown)

/* Do any necessary cleanup:close profiles and dispose of
    color world and bitmap. */
cleanup:
    if (sourceProf)
        CMCloseProfile(sourceProf);
    if (destProf)
        CMCloseProfile(destProf);
    if (cw)
        CWDisposeColorWorld(cw);
    if (bitmap.image)
        DisposePtr(bitmap.image);

    return theErr;
}
```

Embedding Profiles and Profile Identifiers

When the user creates and saves a document or picture containing a color image created or modified with your application, your application can provide for future color matching by saving—along with that document or picture—the profile for the device on which the image was created or modified. In addition to a profile—or instead of a profile—your application can save a profile identifier. A profile identifier is an abbreviated data structure that identifies, and possibly modifies, a profile in memory or on disk.

When embedding source profiles or profile identifiers in the documents created by your application, you can store them in any manner that you choose. For example, you may choose to have your application store, in the resource fork of the document file, one profile for an entire image, or a separate profile for every object in an image, or a separate profile identifier that points to a profile on disk for every device on which the user modified the image.

When embedding source profiles or profile identifiers in PICT file pictures, your application should use the `cmComment` picture comment, which has a `kind` value of 224 and is defined for embedded version 2.x profiles. This comment is followed by a 4-byte selector that describes the type of data in the comment. The following selectors are currently defined:

Selector	Value	Description
`cmBeginProfileSel`	0	Beginning of a version 2.x profile. Profile data to follow.
`cmContinueProfileSel`	1	Continuation of version 2.x profile data. Profile data to follow.
`cmEndProfileSel`	2	End of version 2.x profile data. No profile data follows.
`cmProfileIdentifierSel`	3	Profile identifier follows. A profile identifier identifies a profile that may reside in memory or on disk.

Because the `dataSize` parameter of the `PicComment` procedure is a signed 16-bit value, the maximum amount of profile data that can be embedded in a single picture comment is 32,763 bytes (32,767 – 4 bytes for the selector).

You can embed a larger profile by using multiple picture comments of selector type `cmContinueProfileSel`, as shown in Listing 4-7 (page 67). You must embed the profile data in consecutive order, and you must conclude the profile data by embedding a picture comment of selector type `cmEndProfileSel`. The ColorSync Manager provides the `NCMUseProfileComment` function to automate the process of embedding profile information.

Embedded Profile Format

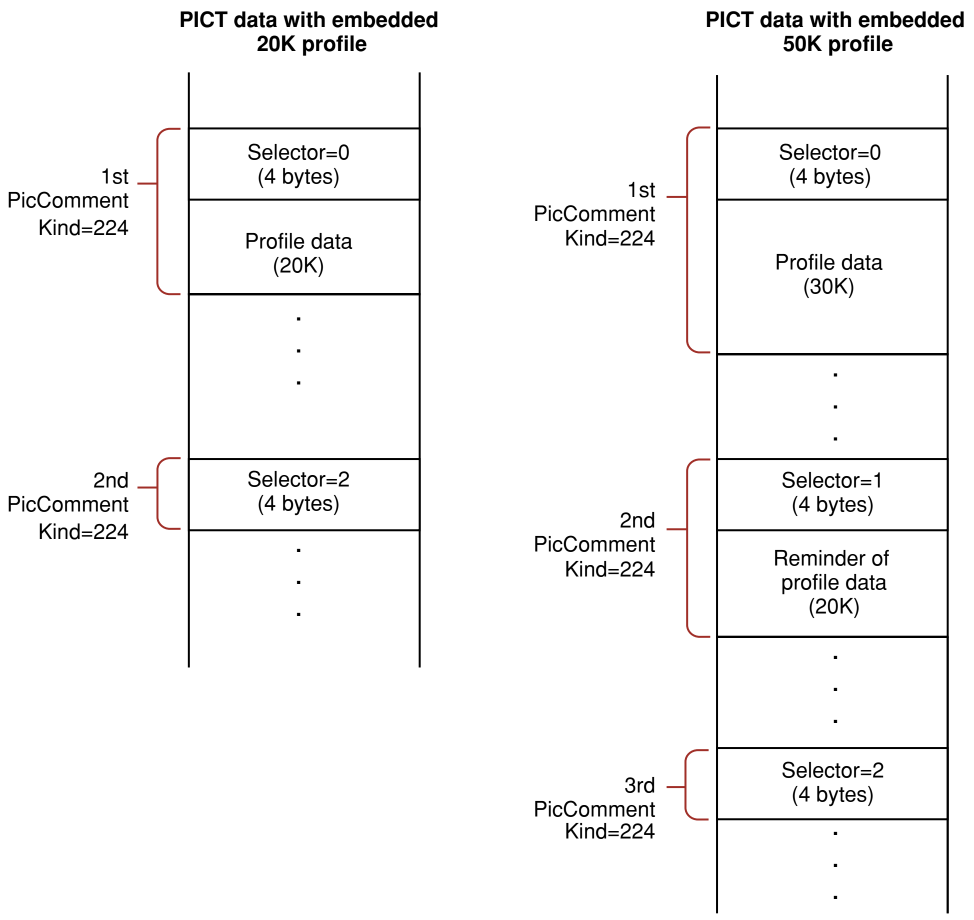
Listing 4-7 (page 67) shows how profile data is embedded in a PICT file picture as a series of picture comments. The illustration shows two embedded profiles. The first profile contains less than 20K of data, so its data can be stored in one picture comment with selector type `cmBeginProfileSel`. Note, however, that a second comment of selector type `cmEndProfileSel`, containing no data, concludes the embedded profile.

The second embedded profile shown in Listing 4-7 (page 67) has more than 32K of data, so its data must be stored in two consecutive picture comments. The first comment has selector type `cmBeginProfileSel`, while the second has type `cmContinueProfileSel`. If the profile were larger and required additional picture comments, each additional comment would have selector type `cmContinueProfileSel`. As with all embedded profiles, the final picture comment has selector type `cmEndProfileSel`.

Embedding Different Profile Versions

For version 1.0 of the ColorSync Manager, you use picture comment types `cmBeginProfile` and `cmEndProfile` to begin and end a picture comment. The `cmBeginProfile` comment is not supported for ColorSync version 2.x profiles; however, the you can use the `cmEndProfile` comment to end the current profile for both ColorSync 1.0 and 2.x. Following a `cmEndProfile` comment, the ColorSync Manager reverts to the system profile. You use the `cmEnableMatching` and `cmDisableMatching` picture comments to begin and end color matching in both ColorSync 1.0 and 2.x. See *Inside Macintosh: Imaging With QuickDraw* for more information about picture comments.

Figure 4-7 Embedding profile data in a PICT file picture



The NCMUseProfileComment Function

The ColorSync Manager provides the function `NCMUseProfileComment` to automate the process of embedding a profile or profile identifier. This function generates the picture comments required to embed the specified profile or identifier into the open picture. It 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: 0 to begin the profile, 1 to continue, and 2 to end the profile; or 3 for a profile identifier. For a profile, 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 1 to embed each segment.

For embedded profiles or profile identifiers to work correctly, the currently effective profile must be terminated by a picture comment of `kindcmEndProfile` after drawing operations using that profile are performed. If you do not specify a picture comment to end the profile, the profile will remain in effect until the next embedded profile is introduced with a picture comment of `kindcmBeginProfile`. It is good practice to 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.

Important: The `NCMUseProfileComment` function does not automatically terminate the embedded profile or profile identifier with a `cmEndProfile` picture comment. You must add a picture comment of `kind cmEndProfile` when the drawing operations to which the profile applies are complete. Otherwise, the profile remains in effect until the next embedded profile with a picture comment of `kind cmBeginProfile` is encountered.

In addition to embedded profiles, an image may contain embedded profile identifiers, which are stored with the selector cmProfileIdentifierSel. For more information on profile identifiers, see Searching for a Profile That Matches a Profile Identifier (page 87), and `CMProfileIdentifier`.

Listing 4-8 (page 72) shows how to embed a profile in a picture file. The `MyPreprendProfileToPicHandle` function creates a new picture, embeds the profile for the device used to create the picture, then draws the picture. The caller passes a reference for the profile as the `prof` parameter. Note that after `MyPreprendProfileToPicHandle` calls the `NCMUseProfileComment` function to embed the profile, it calls its own `MyEndProfileComment` function to embed a comment of `kindcmEndProfile`, ensuring that the profile is properly terminated.

Listing 4-8 Embedding a profile by prepending it before its associated picture

```
CMError MyPrependProfileToPicHandle (
                PicHandle pict,
                PicHandle *pictNew,
                CMProfileRef prof,
                Boolean embedAsIdentifier)
{
    OSErr           theErr;
    CGrafPtr        savePort;
    GDHandle        saveGDev;
    GWorldPtr       tempWorld;
    Rect            pictRect;
    unsigned long   flags;

    // Init for error handling.
    theErr = noErr;
    tempWorld = nil;

    // Check parameters
    if (prof == nil) theErr = paramErr;
    require(theErr == noErr, cleanup);

    // Determine whether to embed as identifier or whole profile.
    if (embedAsIdentifier)
        flags = cmEmbedProfileIdentifier;
    else
        flags = cmEmbedWholeProfile;

    // Create a temporary graphics world.
    theErr = NewSmallGWorld(&tempWorld);
    require(theErr == noErr, cleanup);

    // Save current world and switch to temporary.
    GetGWorld(&savePort, &saveGDev);
    SetGWorld(tempWorld, nil);
    pictRect = (**pict).picFrame;
    ClipRect(&pictRect);                // Important: set clipRgn.
    // Create a new picture.
    *pictNew = OpenPicture(&pictRect);  // Start recording.
    theErr = NCMUseProfileComment(prof,flags);
    DrawPicture(pict, &pictRect);
    MyEndProfileComment();              // Routine shown below.
    ClosePicture();

    if (theErr)
        KillPicture(*pictNew);

    SetGWorld(savePort, saveGDev);

// Do any necessary cleanup:dispose of graphics world.
cleanup:

    if (tempWorld)
        DisposeGWorld(tempWorld);

    return theErr;
}
```

Here is the application-defined `MyEndProfileComment` function called by MyPrependProfileToPicHandle to add the `cmEndProfile` picture comment to terminate the profile:

```
void MyEndProfileComment (void)
{
    PicComment(cmEndProfile, 0, 0);
}
```

Extracting Profiles Embedded in Pictures

To color match or gamut check a picture embedded in a document, your application should first check for embedded profiles in the document. If a profile is found, your application can then open a reference to the profile and use it as the source profile. This process requires you to locate and identify the profile for the image within the document and extract the profile data from the document file.

Note: If you use the QuickDraw-specific `NCMDrawMatchedPicture` function, you do not need to extract the source profile from the PICT file.

To extract an embedded profile, your application can use the function `CMUnflattenProfile`. This function takes a pointer to a low-level data-transfer function that your application supplies to transfer the profile data from the document containing it. This function assumes that your low-level data-transfer function is informed about the context of the profile. After all of the profile data has been transferred, the `CMUnflattenProfile` function returns the file specification for the profile.

Prior to ColorSync 2.5, when your application calls the `CMUnflattenProfile` function, the ColorSync Manager uses the Component Manager to pass the pointer to your low-level data-transfer function along with the reference constant your application can use as it desires. The CMM is determined by the selection process described in How the ColorSync Manager Selects a CMM (page 51). The CMM calls your low-level data-transfer function, directing it to open the file containing the profile, read segments of the profile data, and return the data to the CMM's calling function.

The CMM communicates with your low-level data transfer-function using a command parameter to identify the operation to perform. To facilitate the transfer of profile data from the file to the CMM, the CMM passes to your function a pointer to a data buffer for data, the size in bytes of the profile data your function should return, and the reference constant passed from the calling application.

On return, your function passes to the CMM segments of the profile data and the number of bytes of profile data you actually return.

Starting with ColorSync 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM. On return from `CMUnflattenProfile`, the value of `preferredCMMnotfound` is guaranteed to be `false`.

Listing 4-9 (page 74) and Listing 4-10 (page 76) show portions of a sample application called CSDemo, available as part of the ColorSync SDK. You can find the complete sample application on the Developer CD series, or at the web site <http://developer.apple.com/sdk>.

In these listings, all variables beginning with a lowercase letter g are global variables previously defined. The application uses global variables to pass data between functions that do not include reference constant parameters. Listing 4-9 (page 74) counts the profiles in a PICT file, while Listing 4-10 (page 76) extracts a profile, identified by an index number, from a PICT file.

Counting the Profiles in the PICT File

Given a `picHandle` value to a picture containing an embedded profile, the sample code shown in Listing 4-10 (page 76) counts the number of profiles in the picture.

The `MyCountProfilesInPicHandle` function calls the Toolbox function SetStdCProcs to get the current QuickDraw drawing bottleneck procedures, then sets the bottlenecks to its own routines. It initializes its global counter, `gCount`, which holds a single count summing both ColorSync 1.0 profiles and version 2.x profiles, to zero. The `MyCountProfilesInPicHandle` function calls its own drawing function, `MyDrawPicHandleUsingBottleneck`, not shown here, to draw the picture. The drawing function sets up a port that uses the private bottleneck routines.

As the picture is drawn, the MyCountProfilesCommentProc bottleneck procedure counts the number of profiles encountered. `MyCountProfilesCommentProc` checks for both version 1.0 profiles and version 2.x profiles and increments the global count when it finds either type. You can easily modify this code to keep separate counts if necessary.

`MyCountProfilesInPicHandle` doesn't use any other QuickDraw bottlenecks, so it uses nonoperational routines (routines that do nothing but return) for all other bottlenecks. The prototype for a function to handle the `TextProc` bottleneck, for example, can be defined as follows:

```
static pascal void MyNoOpTextProc ( short byteCount,
                                    Ptr textAddr,
                                    Point numer,
                                    Point denom);
```

For a general discussion of customizing QuickDraw's bottleneck routines, see Customizing QuickDraw's Text Handling in Inside Macintosh: Text.

Listing 4-9 Counting the number of profiles in a picture

```
CMError MyCountProfilesInPicHandle (PicHandle pict, unsigned long *count)
{
    OSErr       theErr = noErr;
    CQDProcs    procs;

    /* Set up bottleneck for picComments so we can count the profiles. */
    SetStdCProcs(&procs);
    procs.textProc    = NewQDTextProc (MyNoOpTextProc);
    procs.lineProc    = NewQDLineProc (MyNoOpLineProc);
    procs.rectProc    = NewQDRectProc (MyNoOpRectProc);
    procs.rRectProc   = NewQDRRectProc (MyNoOpRRectProc);
    procs.ovalProc    = NewQDOvalProc (MyNoOpOvalProc);
    procs.arcProc     = NewQDArcProc (MyNoOpArcProc);
    procs.polyProc    = NewQDPolyProc (MyNoOpPolyProc);
    procs.rgnProc     = NewQDRgnProc (MyNoOpRgnProc);
    procs.bitsProc    = NewQDBitsProc (MyNoOpBitsProc);
    procs.commentProc = NewQDCommentProc(MyCountProfilesCommentProc);
    procs.txMeasProc  = NewQDTxMeasProc (MyNoOpTxMeasProc);

/* Initialize the global counter to be incremented by the commentProc. */
    gCount = 0;

/* Draw the picture and count the profiles while drawing. */
    theErr = MyDrawPicHandleUsingBottlenecks (pict, procs, nil);

/* Obtain the result from the count global variable. */
    *count = gCount;

/* Clean up and return. */
    DisposeRoutineDescriptor(procs.textProc);
    DisposeRoutineDescriptor(proc.lineProc);
    DisposeRoutineDescriptor(procs.rectProc);
    DisposeRoutineDescriptor(procs.rRectProc);
    DisposeRoutineDescriptor(procs.ovalProc);
    DisposeRoutineDescriptor(procs.arcProc);
    DisposeRoutineDescriptor(procs.polyProc);
    DisposeRoutineDescriptor(procs.rgnProc);
    DisposeRoutineDescriptor(procs.bitsProc);
    DisposeRoutineDescriptor(procs.commentProc);
    DisposeRoutineDescriptor(procs.txMeasProc);
}

pascal void MyCountProfilesCommentProc (short kind,
                                        short dataSize,
                                        Handle dataHandle)
{
    long    selector;

    switch (kind)
    {
        case cmBeginProfile
           gCount ++;  // We found a ColorSync 1.0 profile; increment the count.
            break;

        case cmComment;
            // Break if dataSize is too small to be a selector.
            if (dataSize <= 4) break;

            // Since dataSize is >= 4, we can get a selector from the first
long.
```

```
            selector = *((long *)(*dataHandle));
            if (selector == cmBeginProfileSel)
                gCount ++;  // We found a ColorSync 2.xprofile; increment the
count.
            break;
    }
}
```

Extracting a Profile

Flattening refers to transferring a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document. Unflattening refers to transferring from the embedded format to an independent disk file.

This part of the sample application identifies the profile to unflatten, unflattens the profile, creates a temporary profile, and disposes of the original. To perform these tasks, the code must again draw the picture using the bottleneck routines.

Part A: Calling the Unflatten Function

Listing 4-10 (page 76) shows the MyGetIndexedProfileFromPicHandle entry point function that drives the process of unflattening the profile. The function creates a universal procedure pointer (UPP), `MyflattenUPP`, that points to the low-level data-transfer procedure.

A PICT handle may contain more than one profile. To identify the profile to unflatten, the `MyGetIndexedProfileFromPicHandle` function contains an `index` parameter that specifies the profile's index. The function stores the index in the global variable `gIndex` so that the value is accessible by the application's other functions that check for the correct profile and extract it. Then, the function calls the `CMUnflattenProfile` function, passing it the `MyflattenUPP` pointer. This invokes the `MyUnflattenProc` function shown in Listing 4-11 (page 78).

The function MyGetIndexedProfileFromPicHandle, shown in Listing 4-10 (page 76), first calls `CMUnflattenProfile` to create an independent file-based profile, then calls the function `CMOpenProfile` to open a temporary profile reference to the file-based profile. It then calls `CMCopyProfile` to create a copy of the profile reference. Finally, the function disposes of the original profile. The purpose for creating a temporary profile, copying it into the specified location, then deleting the temporary profile, is to adhere to the copyright protection for embedded profiles specified by the `flags` field in the profile header.

Listing 4-10 Calling the CMUnflattenProfile function to extract an embedded profile

```
CMError MyGetIndexedProfileFromPicHandle (PicHandle pict,
                                            unsigned long index,
                                            CMProfileRef *prof,
                                            CMProfileLocation *profLoc)
{
    CMError             theErr;
    unsigned long       refCon;
    CMFlattenUPP        myFlattenUPP;
    Boolean             preferredCMMNotFound;
    Boolean             tempCreated;
    FSSpec              tempSpec;
    CMProfileRef        tempProf;
    CMProfileLocation   tempProfLoc;
    // Init for error handling.
    theErr = noErr;
    tempCreated = false;
    tempProf = nil;

    // Create a universal procedure pointer for the
    // unflatten procedure shown in Listing 3-11.
    myFlattenUPP = NewCMFlattenProc(MyUnflattenProc);

    // Pass the pict as the refcon.
    refCon = (unsigned long) pict;

    // Set the global index variable to the index of the profile we're looking
 for.
    gIndex = index;

    // The next call invokes the MyUnflattenProc shown in Listing 3-11.
    //On return, tempSpec identifies the newly created profile on disk.
    theErr = CMUnflattenProfile(&tempSpec, myFlattenUPP,(void*)&refCon,
                            &preferredCMMNotFound);
   DisposeRoutineDescriptor(myFlattenUPP);// Dispose of the procedure pointer.
    require(theErr == noErr, cleanup);
    tempCreated = true;

    // Open the newly created profile, create a temporary profile reference for
 it,
    // copy the temporary reference, then close it and delete the profile file.
    tempProfLoc.locType = cmFileBasedProfile;
    tempProfLoc.u.fileLoc.spec = tempSpec;

    theErr = CMOpenProfile(&tempProf, &tempProfLoc);
    require(theErr == noErr, cleanup);

    theErr = CMCopyProfile(prof, profLoc, tempProf);
    require(theErr == noErr, cleanup);

// Do any necessary cleanup:close profile and delete file spec.
cleanup:

    if (tempProf)
        theErr = CMCloseProfile(tempProf);

    if (tempCreated)
        theErr = FSpDelete(&tempSpec);

    return theErr;
}
```

Part B: Unflattening the Profile

Prior to ColorSync 2.5, your transfer function is called by the CMM that handles the unflatten operation. Starting with ColorSync 2.5, however, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM.

When the code in MyGetIndexedProfileFromPicHandle (Listing 4-10 (page 76)) calls the `CMUnflattenProc` function, passing it a pointer to the `MyUnflattenProc` function, the `MyUnflattenProc` function (Listing 4-11 (page 78)) is called by ColorSync or by the CMM (depending on the version of ColorSync) to perform the low-level profile data transfer from the document file.

When the `MyUnflattenProc` function is called with an open command, the function initializes global variables, creates a graphics world, and installs bottleneck procedures in the graphics world. The only bottleneck procedure actually used is `MyUnflattenProfilesCommentProc`, which checks the picture comments as the picture is drawn offscreen to identify the desired profile. For a general discussion of customizing QuickDraw's bottleneck routines, see Customizing QuickDraw's Text Handling in Inside Macintosh: Text.

When the `MyUnflattenProc` function is called with a read command, the function reads the appropriate segment of data from a chunk and returns it. To accomplish this, it calls the MyDrawPicHandleUsingBottlenecks function with the appropriate bottleneck procedure installed. In turn, this invokes the `MyUnflattenProfilesCommentProc` shown in Listing 4-12 (page 80).

When the `MyUnflattenProc` function is called with a close command, the function releases any memory it allocated and disposes of the graphics world and bottlenecks.

Listing 4-11 The unflatten procedure

```
pascal OSErr MyUnflattenProc (long command,
                                long *sizePtr,
                                void *dataPtr,
                                void *refConPtr)
{
    OSErr                   theErr = noErr;
    static CQDProcs         procs;
    static GWorldPtr        offscreen;
    PicHandle               pict;
    switch (command)
    {
        case cmOpenReadSpool:
            theErr = NewSmallGWorld(&offscreen);
            if (theErr)
                return theErr;

            /* Replace the QuickDraw bottleneck routines, mostly with routines
                that do nothing, but also with our unflatten comments routine,
                so that we can intercept the comments we are interested in and
                ignore everything else. */
            SetStdCProcs(&procs);
            procs.textProc    = NewQDTextProc (MyNoOpTextProc);
            procs.lineProc    = NewQDLineProc (MyNoOpLineProc);
            procs.rectProc    = NewQDRectProc (MyNoOpRectProc);
            procs.rRectProc   = NewQDRRectPro (MyNoOpRRectProc);
            procs.ovalProc    = NewQDOvalProc (MyNoOpOvalProc);
            procs.arcProc     = NewQDArcProc (MyNoOpArcProc);
            procs.polyProc    = NewQDPolyProc (MyNoOpPolyProc);
            procs.rgnProc     = NewQDRgnProc (MyNoOpRgnProc);
            procs.bitsProc    = NewQDBitsProc(MyNoOpBitsProc);
          procs.commentProc = NewQDCommentProc (MyUnflattenProfilesCommentProc);
            procs.txMeasProc  = NewQDTxMeasProc (MyNoOpTxMeasProc);

            gChunkBaseHndl = nil;
            gChunkIndex = 0;
            gChunkOffset = 0;
            gChunkSize = 0;
            break;

    case cmReadSpool:
        if (gChunkOffset > gChunkSize)      /* If we overread the last chunk,
*/
        {
            return ioErr;                   /* use system I/O error value. */
        }
        if (gChunkOffset == gChunkSize)     /* If we used up the last chunk, */
        {
            if (gChunkBaseHndl !=nil)
            {
                HUnlock(gChunkBaseHndl);    /* dispose of the previous chunk.
*/
                DisposeHandle(gChunkBaseHndl);
                gChunkBaseHndl = nil;
            }
            gChunkIndex++;              /* Read in a new chunk. */
            gChunkOffset = 0;
            gCount = 0;
            gChunkCount = 0;
            pict = *((PicHandle *)refConPtr);
            theErr = MyDrawPicHandleUsingBottlenecks (pict, procs, offscreen);
            /* This invokes MyUnflattenProfilesCommentProc shown in Listing
3-12. */
            if (gChunkBaseHndl==nil)    /* Check to see if we're overread. */
               return ioErr;          /* If so, return system I/O error value.
 */
            HLock(gChunkBaseHndl);
        }
        if (gChunkOffset < gChunkSize)
        {
            *sizePtr = MIN(gChunkSize-gChunkOffset, *sizePtr);
            BlockMove((Ptr)(&((*gChunkBaseHndl)[gChunkOffset])),
                (Ptr)dataPtr, *sizePtr);
            gChunkOffset += (*sizePtr);
        }
        break;
        case cmCloseSpool:
            if (gChunkBaseHndl != nil)
            {
              HUnlock(gChunkBaseHndl);       /* Dispose of the previous chunk.
 */
                DisposeHandle(gChunkBaseHndl);
                gChunkBaseHndl = nil;
            }
            /* Dispose of our offscreen world and the routine descriptors
                for our bottlenect routines. */
            DisposeGWorld(offscreen);
            DisposeRoutineDescriptor(procs.MyNoOpTextPrc);
            DisposeRoutineDescriptor(procs.MyNoOpLinePrc);
            DisposeRoutineDescriptor(procs.MyNoOpRectProc);
            DisposeRoutineDescriptor(procs.MyNoOpRRectPrc);
            DisposeRoutineDescriptor(procs.MyNoOpOvalProc);
            DisposeRoutineDescriptor(procs.MyNoOpArcProc);
            DisposeRoutineDescriptor(procs.MyNoOpPolyPrc);
            DisposeRoutineDescriptor(procs.MyNoOpRgnProc);
            DisposeRoutineDescriptor(procs.MyNoOpBitsProc);
            DisposeRoutineDescriptor(procs.MyUnflattenProfilesCommentPrc);
            DisposeRoutineDescriptor(procs.MyNoOpTxMeasPrc);
            break;
        default:
            break;
    }
    return theErr;
}
```

Part C: Calling the Comment Procedure

When the `MyUnflattenProc` function's `MyDrawPicHandleUsingBottlenecks` function calls the `MyUnflattenProfilesCommentProc` function, the function shown in Listing 4-12 (page 80) finds the profile identified by the index, finds the correct segment of data within the profile, and stores the data in the `gChunkBaseHndl` global variable.

Listing 4-12 The comment procedure

```
pascal void MyUnflattenProfilesCommentProc (short kind,
                                            short dataSize,
                                            Handle dataHandle)
{
    long    selector;
    OSErr   theErr;

    if (gChunkBaseHndl != nil) return;
                /* The handle is in use; this shouldn't happen. */
    if (gCount > gIndex) return;
                /* We have already found the profile. */
    switch (kind)
    {
    case cmBeginProfile:
        gCount ++;          /* We found a version 1 profile. */
        gChunkCount = 1;    /* v1 profiles should only have 1 chunk. */
        if (gCount != gIndex) break;
                            /* This is not the profile we're looking for. */
        if (gChunkCount != gChunkIndex) break;
                            /* This is not the chunk we're looking for. */
        gChunkBaseHndl = dataHandle;
        theErr = HandToHand(&gChunkBaseHndl);
        gChunkSize = dataSize;
        gChunkOffset = 0;
        break;

    case cmComment:
        if (dataSize <= 4) break;
                            /* The dataSize too small for selector, so break.
*/
        selector = *((long *)(*dataHandle));
                            /* Get the selector from the first long in data. */
        switch (selector)
        {
            case cmBeginProfileSel:
                gCount ++;              /* We found a version 2 profile. */
                gChunkCount = 1;
                if (gCount != gIndex) break;
                                    /* This is not the profile we're looking
for. */
                if (gChunkCount!=gChunkIndex) break;
                                   /* This is not the chunk we're looking for.
 */
                gChunkBaseHndl = dataHandle;
                theErr = HandToHand(&gChunkBaseHndl);
                gChunkSize = dataSize;
                gChunkOffset = 4;
                break;

            case cmContinueProfileSel:
                gChunkCount ++;
                if (gCount != gIndex) break;
                                    /* This is not the profile we're looking
for. */
                if (gChunkCount!=gChunkIndex) break;
                                   /* This is not the chunk we're looking for.
 */
                gChunkBaseHndl = dataHandle;
                theErr = HandToHand(&gChunkBaseHndl);
                gChunkSize = dataSize;
                gChunkOffset = 4;
                break;

            case cmEndProfileSel:
                                    /* Check to see if we're overreading. */
                gChunkCount = 0;
                break;
        }
        break;
    }
}
```

Performing Optimized Profile Searching

Starting with version 2.5, ColorSync provides a profile cache and a new routine, `CMIterateColorSyncFolder`, for optimized profile searching. The sample code shown in Listing 4-13 (page 82) through Listing 4-15 (page 85) takes advantage of optimized searching if ColorSync version 2.5 is available; if not, it performs a search that is compatible with earlier versions of ColorSync. The compatible search may take some advantage of the profile cache, but cannot provide fully optimized results.

As background for the code samples in Listing 4-13 (page 82) to Listing 4-15 (page 85), you should be familiar with the topics described in the following sections:

- Profile Location (page 32)
- Profile Search Locations (page 34)
- The Profile Cache and Optimized Searching (page 35)

Important: You cannot use the ColorSync Manager search functions to search for ColorSync 1.0 profiles.

The `CMIterateColorSyncFolder` function uses ColorSync's profile cache to supply your application with information about the profiles currently available in the ColorSync Profiles folder. The function calls your callback routine once for each available profile, supplying your routine with the profile header, script code, name, and location, stored in a structure of type `CMProfileIterateData`.

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. 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, quickly and without having to open each file-based profile.

An Iteration Function for Profile Searching With ColorSync 2.5

The CMIterateColorSyncFolderCompat function, shown in Listing 4-15 (page 85), performs an optimized search using the `CMIterateColorSyncFolder` function if ColorSync version 2.5 is available. Otherwise, it calls `theCMNewProfileSearch` function, which is available in earlier versions of ColorSync.

When you call the CMIterateColorSyncFolderCompat function, you pass a universal procedure pointer to a filter procedure in the `proc` parameter. CMIterateColorSyncFolderCompat uses that filter procedure when it performs an optimized search with `CMIterateColorSyncFolder`. Listing 4-13 (page 82) provides a sample filter procedure called `MyIterateProc`.

The `MyIterateProc` function is called once for each available profile and merely stores the names of all non-display profiles (such as printer and scanner profiles) at an arbitrary position in a list. You would do something similar, for example, to display a list of profiles in a dialog.

Note that the CMIterateColorSyncFolderCompat function works in a similar way for ColorSync 2.5 and for earlier versions, although the search is much more efficient with version 2.5. CMIterateColorSyncFolderCompat either calls the `CMIterateColorSyncFolder` function, which calls the MyIterateProc function once for each available profile, or it calls the `CMNewProfileSearch` function, which calls the ProfileSearchFilter function (Listing 4-14 (page 83)) once for each available profile. The ProfileSearchFilter function in turn calls MyIterateProc, so similar processing occurs.

Listing 4-13 An iteration function for profile searching with ColorSync 2.5

```
Pascal OSErr MyIterateProc(CMProfileIterateData* data, void* refcon)
{
    Cell theCell;
    // Assume we can cast refCon to a ListHandle.
    ListHandle list = (ListHandle)refcon;

    /* Assume we're interested only in non-display profiles, such as printer
        and scanner profiles. */
    if (data->header.profileClass != cmDisplayClass)
    {
        /* This code adds the profile name at an arbitrary position
            in a list. You could do something similar to display a
            list of all available profiles. */
        cell.v = LAddRow(1,999,list);
        cell.h = 0;
        // The name data in the iterate data structure is in Pascal format,
            so we use the length byte to determine how many bytes to copy. */
        LSetCell((Ptr)data->name+1, name[0], cell, list);
        cell.h = 1;
        // Store the profile's location information with the cell.
        LSetCell((Ptr)data->location, sizeof(cmProfileLocation), cell, list);
    }
    // A more complicated function might need to return an error here.
    return noErr;
};
```

A Filter Function for Profile Searching Prior to ColorSync 2.5

To search for profiles prior to version 2.5 of the ColorSync Manager, you use the `CMNewProfileSearch` function. You supply `CMNewProfileSearch` with a search record of type `CMSearchRecord` that identifies the search criteria. If you also provide a pointer to a filter function, `CMNewProfileSearch` uses the function to eliminate profiles from the search based on additional criteria not defined by the search record. The `ProfileSearchFilter` function shown in Listing 4-14 (page 83) provides an example of a filter routine for searching with the `CMNewProfileSearch` function.

Listing 4-14 (page 83) defines the IterateCompatPtr data type, a pointer to a structure that stores search information. When you call the CMIterateColorSyncFolderCompat function shown in Listing 4-15 (page 85), you pass a reference to the MyIterateProc function (Listing 4-13 (page 82)) in the proc parameter. If ColorSync 2.5 is not available, the CMIterateColorSyncFolderCompat function calls the `CMNewProfileSearch` function. It passes the `ProfileSearchFilter` function (Listing 4-13 (page 82)) as the search filter and it passes an IterateCompatPtr pointer as the `refCon` parameter. It sets the `proc` field of the IterateCompatPtr pointer to the MyIterateProc function that you passed in the `proc` parameter.

The `CMNewProfileSearch` function calls the `ProfileSearchFilter` function (Listing 4-13 (page 82)) once for each profile. The `ProfileSearchFilter` function simply casts the passed `refCon` pointer to an IterateCompatPtr, then calls the function specified by the pointer's proc field. As a result, the MyIterateProc function is called once for each profile, just as it is when CMIterateColorSyncFolderCompat calls CMIterateColorSyncFolder under ColorSync 2.5.

Note that `ProfileSearchFilter` always returns true, indicating the profile should be filtered out of the search result returned by `CMNewProfileSearch`, because we've already gotten all the information we need from it. Note also that `ProfileSearchFilter` uses the `require` macro, which is defined in Poor Man's Exception Handling (page 60).

Listing 4-14 A filter function for profile searching prior to ColorSync 2.5

```
// Declare a structure to use for searching with ColorSync versions prior to
2.5.
typedef struct IterateCompat
{
    CMProfileIterateUPP     proc;
    OSErr                   osErr;
    void*                   refCon;
} IterateCompatRec, *IterateCompatPtr;

static pascal Boolean ProfileSearchFilter (CMProfileRef prof, void *refCon)
{
    OSErr                   theErr = noErr;
    IterateCompatPtr        refConCompatPtr;
    CMProfileIterateData    iterData;

    // Cast refcon to our type.
    refConCompatPtr = (IterateCompatPtr)refCon;
    // If we had an error from an earlier profile, give up
    //  by branching to cleanup location.
    theErr = refConCompatPtr->osErr;
    require(theErr == noErr, cleanup);

    // Try to get the profile's location.
    theErr = CMGetProfileLocation(prof, &iterData.location);
    require(theErr == noErr, cleanup);

    // Try to get the profile's header.
    theErr = CMGetProfileHeader(prof, (CMAppleProfileHeader*)&iterData.header);
    require(theErr == noErr, cleanup);

    // Try to get the profile's name.
    theErr = CMGetScriptProfileDescription(prof, iterData.name, &iterData.code);
    require(theErr == noErr, cleanup);

    iterData.dataVersion = cmProfileIterateDataVersion1;

    // Call the iterate callback routine.
    theErr = CallCMProfileIterateProc(refConCompatPtr->proc,
                                &iterData, refConCompatPtr->refCon);
    require(theErr == noErr, cleanup);

cleanup:

    if (theErr)
        refConCompatPtr->osErr = theErr;

    return true;    // exclude the profile;
}
```

A Compatible Function for Optimized Profile Searching

Listing 4-15 (page 85) provides sample code that performs an optimized profile search if ColorSync 2.5 is available, but provides a search that is compatible with previous versions if ColorSync 2.5 is not available.

When ColorSync 2.5 is available, CMIterateColorSyncFolderCompat simply calls the function `CMIterateColorSyncFolder`, passing on the information it received through its parameters. As a result, `CMIterateColorSyncFolder` calls the MyIterateProc function, shown in Listing 4-13 (page 82), once for each available profile. Your version of MyIterateProc can examine the passed information for each profile and perform any required operation on the profiles it is interested in.

When ColorSync 2.5 is not available, CMIterateColorSyncFolderCompat sets up a search with the function `CMNewProfileSearch`. As part of this setup, it initializes a structure of type IterateCompatRec, defined in Listing 4-14 (page 83), which it passes to `CMNewProfileSearch` for the `refCon` parameter. The `CMNewProfileSearch` function in turn passes a pointer to the IterateCompatRec structure as the `refCon` parameter to ProfileSearchFilter, which it calls once for each available profile.

ProfileSearchFilter calls the MyIterateProc function, which gets a chance to handle each profile, just as it does in the case where ColorSync 2.5 is available. The main drawback is that without the availability of the profile cache and the `CMIterateColorSyncFolder` function, searching through the profiles is likely to be a much more time-consuming task.

Note that CMIterateColorSyncFolderCompat uses the `require` macro, which is defined in Poor Man's Exception Handling (page 60).

Listing 4-15 Optimized profile searching compatible with previous versions of ColorSync

```
CMError CMIterateColorSyncFolderCompat (CMProfileIterateUPP proc,
                                         unsigned long *seed,
                                         unsigned long *count,
                                         void *refCon)
{
    CMError theErr = noErr ;

    /* Presume the caller passed a pointer to MyIterateProc to this
            function in the proc parameter. */
    if ( ColorSync25Available() )
        return CMIterateColorSyncFolder(proc, seed, count, refCon);
    else
    {
        CMProfileSearchRef  searchResult;
        CMSearchRecord      searchSpec;
        unsigned long       count;
        IterateCompatRec    refConCompat;


        searchSpec.filter = NewCMProfileFilterProc(ProfileSearchFilter);
        searchSpec.searchMask = cmMatchAnyProfile;

        /* Set up our private data structure for compatible (pre-ColorSync 2.5)
            profile searching.
            Pass the pointer to the MyIterateProc function, which was
                presumably passed to this function in the proc parameter,
                on to our filter routine, ProfileSearchFilter,
                in the refCon parameter, using an IterateCompatRec structure.
*/
        refConCompat.proc = proc;
        refConCompat.osErr = noErr;
        refConCompat.refCon = refCon;

        //  Start traditional search.
        theErr = CMNewProfileSearch(&searchSpec,
                            (void*)&refConCompat, &count, &searchResult);
        if (theErr == noErr)
        {
            // We don't use the result, but still must dispose of it.
            CMDisposeProfileSearch(searchResult);
            theErr = refConCompat.osErr;
        }
        DisposeRoutineDescriptor(searchSpec.filter);
    }

    return theErr;
}
```

Searching for Specific Profiles Prior to ColorSync 2.5

Starting with version 2.5, you can do fast, optimized profile searching that takes advantage of the profile cache added in ColorSync 2.5. For an overview, see The Profile Cache and Optimized Searching (page 35). The sample code in Listing 4-15 (page 85) takes advantage of optimized searching if ColorSync version 2.5 is available; if not, it performs a search that is compatible with earlier versions of ColorSync. The compatible search may take some advantage of the profile cache, but cannot provide fully optimized results.

Listing 4-16 (page 86), shown in this section, provides an additional example of the searching mechanism available prior to ColorSync version 2.5.

Important: You cannot use the ColorSync Manager search functions to search for ColorSync 1.0 profiles.

Your application can use the ColorSync Manager search functions to obtain a list of profiles in the ColorSync Profiles folder that meet specifications you supply in a search record. For example, you can use these functions to find all profiles for printers that meet certain criteria defined in the profile. Your application can walk through the resulting list of profiles and obtain the name and script code of each profile corresponding to a specific index in the list. Your application can then display a selection menu showing the names of the profiles. Listing 4-16 (page 86) shows sample code that takes an approach similar to the one this example describes.

Note: You can also search the ColorSync Profiles folder for profiles that match a profile identifier. For more information, see Searching for a Profile That Matches a Profile Identifier (page 87), and `CMProfileIdentifierFolderSearch`.

The MyProfileSearch function, shown in Listing 4-16 (page 86), defines values for the search specification record fields, including the search mask, and assigns those values to the record's fields after initializing the search result. Then MyProfileSearch calls the CMNewProfileSearch function to search the ColorSync Profiles folder for profiles that meet the search specification requirements. The `CMNewProfileSearch` function returns a one-based count of the profiles matching the search specification and a reference to the search result list of the matching profiles.

Next the MyProfileSearch function calls the `CMSearchGetIndProfile` function to obtain a reference to a specific profile corresponding to a specific index into the search result list. Passing the profile reference returned by the `CMSearchGetIndProfile` function as the `foundProf` parameter, MyProfileSearch calls the `CMGetScriptProfileDescription` function to obtain the profile name and script code.

Finally, the MyProfileSearch function cleans up, calling the `CMCloseProfile` function to close the profile and the `CMDisposeProfileSearch` function to dispose of the search result list.

Listing 4-16 Searching for specific profiles in the ColorSync Profiles folder

```
/* field definitions for search */
#define kCMMType        'appl'          /* ColorSync default CMM */
#define kProfileClass   cmDisplayClass  /* monitor */
#define kAttr0          0x00000000
#define kAttr1          0x00000002      /* Macintosh standard gamma */

/* Define mask to search for profiles that match on CMM type, profile class,
        and attributes. */
```

```
#define kSearchMask (cmMatchProfileCMMType + cmMatchProfileClass +
cmMatchAttributes)
void MyProfileSearch (void)
{
    CMError             cmErr;
    CMProfileRef        foundProf;
    Str255              profName;
    ScriptCode          profScript;
    CMSearchRecord      searchSpec;
    CMProfileSearchRef  searchResult;
    unsigned long       searchCount;
    unsigned long       i;
    /* Init for error handling. */
    searchResult = NULL;
    /* Specify search. */
    searchSpec.CMMType = kCMMType;
    searchSpec.profileClass = kProfileClass;
    searchSpec.deviceAttributes[0 ]= kAttr0;
    searchSpec.deviceAttributes[1] = kAttr1;

    searchSpec.searchMask = kSearchMask;
    searchSpec.filter= NULL;                /* Filter proc is not used. */
    cmErr = CMNewProfileSearch(&searchSpec, NULL, &searchCount, &searchResult);
    if (cmErr == noErr)
    {
        for (i = 1; i <= searchCount; i++)
        {
            if (CMSearchGetIndProfile(searchResult, i, &foundProf) != noErr)
            {
                break;
            }

            cmErr = CMGetScriptProfileDescription(foundProf, profName,
&profScript);
            if (cmErr == noErr)
            {
                /* Assume profile name ScriptCode is smRoman. */
                (void) printf("%s\n", p2cstr(profName));
            }

            (void) CMCloseProfile(foundProf);
        }
    }
    if (searchResult != NULL)
    {
        CMDisposeProfileSearch(searchResult);
    }
}
```

Searching for a Profile That Matches a Profile Identifier

Embedding a profile in an image guarantees that the image can be rendered correctly on a different system. However, profiles can be large—the largest can be more than 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 flag values in an image without having to embed an entire copy of a profile. For more information on the profile identifier structure, including a description of how a match is determined between a profile reference and a profile identifier, see `CMProfileIdentifier`.

Important: A document containing an embedded profile identifier can not necessarily be ported to different systems or platforms.

The ColorSync Manager provides the `NCMUseProfileComment` routine to embed profiles and profile identifiers in an open picture file. For information on embedding, see Embedding Profiles and Profile Identifiers (page 69). 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 the `CMProfileIdentifierListSearch` routine for finding a profile identifier in a list of profile identifiers and the `CMProfileIdentifierFolderSearch` routine for finding a profile identifier in the ColorSync Profiles folder.

When your application or device driver processes an image, it typically keeps a list of profile references for each profile it encounters in the image. Each time it encounters an embedded profile identifier, your application first calls the CMProfileIdentifierListSearch function to see if there is already a matching profile reference in its list. That function returns a list of profile references that match the profile identifier. Although the returned list would normally contain at most one reference, it is possible to have two or more matches. If the CMProfileIdentifierListSearch routine does not find a matching profile reference, your application calls the CMProfileIdentifierFolderSearch routine to see if a matching profile can be found in the ColorSync Profiles folder.

Listing 4-17 (page 88) demonstrates how your application can use the ColorSync Manager's search routines to obtain a profile reference for an embedded profile identifier. It uses the following structure to store a list of profile identifiers, along with a count of the number of items in the list.

```
typedef struct {
    long count;
    CMProfileRef profs[1];
} ProfileCacheList, **ProfileCacheHandle;
```

Listing 4-17 Searching for a profile that matches a profile identifier

```
CMError MyFindAndOpenProfileByIdentifier(ProfileCacheHandle profCache,
                                        CMProfileIdentifierPtr unique,
                                        Boolean *pFoundInCache,
                                        CMProfileRef *pProf)
{
    CMError         theErr = noErr;
    CMProfileRef    prof = nil;
    long            cacheCount = (**profCache).count;
    unsigned long   foundCount = 0;

    *pFoundInCache = false;

    /* If there are any profile references in the cache (the list of profile
        references for profiles or profile identifiers we have already
        encountered) look there for a match with the passed profile identifier.
 */
```

```
    if (cacheCount)
    {
        CMProfileRef *cacheList;

        cacheList = (**profCache).profs;
        foundCount = 1; // return no more than one match
        theErr = CMProfileIdentifierListSearch(unique, cacheList, cacheCount,
                                            &foundCount, &prof);
        if (foundCount && !theErr)
            *pFoundInCache = true;
        else
            prof = nil;
    }

    /* If we didn't find a match for the passed profile identifier in the list
 of
        previously encountered profiles, look for a match on disk, in the
        ColorSync Profiles folder */
    if (!prof)
    {
        CMProfileSearchRef search = nil;
        foundCount = 0;

       theErr = CMProfileIdentifierFolderSearch(unique, &foundCount, &search);
        /* If we found one or more matches, obtain a profile reference for the
            first matching profile; if no error, dispose of the search result.
 */
        if (!theErr)
        {
            if (foundCount)
                theErr = CMSearchGetIndProfile(search, 1, &prof);
            CMDisposeProfileSearch(search);
        }
    }

    /* If we still didn't find a match for the passed profile identifier,
        use the system profile. */
    if (!prof)
    {
        theErr = CMGetSystemProfile(&prof);
    }

    if (theErr)
        prof = nil;
    *pProf = prof;
    return theErr;
}
```

Although typically there is at most one profile reference in your application's list or one profile in the ColorSync Profiles folder that matches the searched-for profile identifier, it is possible that two or more profiles may qualify. It is not an error condition if either the CMProfileIdentifierListSearch or the CMProfileIdentifierFolderSearch routine finds no matching profile.

Checking Colors Against a Destination Device's Gamut

Different imaging devices (scanners, displays, printers) work in different color spaces, and each can have a different gamut or range of colors that they can produce. The process of matching colors between devices entails adjusting the colors of an image from the color gamut of one device to the color gamut of another device so that the resulting image looks as similar as possible to the original image. Not all colors can be rendered on all devices. The rendering intent used in the color transformation process dictates how the colors are matched, strongly influencing the outcome. Your application can give a user some control over the outcome by allowing the user to select the rendering intent. However, some users might want to know in advance which colors are out of gamut for the destination device so that they can choose other appropriate colors within the gamut.

Using the ColorSync Manager general purpose color-checking functions, your application can check the colors of a pixel map (using the `CWCheckPixMap` function), the colors of a bitmap (using the `CWCheckBitMap` function), or a list of colors (using the `CWCheckColors` function) against the color gamut of the destination device and provide a warning when a color is out of gamut for that device.

There are a number of ways in which your application can provide gamut-checking services. For example, you can use gamut checking to see if a given color is reproducible on a particular printer. If the color is not directly reproducible—that is, if it is out of gamut—you could alert the user to that fact.

You can allow a user to specify a list of colors that fall within the gamut of a source device to see if they fit within the gamut of a destination device before the user color matches an image. Your application could display the results in a window, indicating which colors are in the gamut and which are out. This feature, too, gives the user the opportunity to test colors and select different ones for portions of an image whose colors fall out of gamut. To handle this feature, your application can call the `CWCheckColors` function.

In addition to providing features that allow a user to anticipate which colors are out of gamut for a particular device, your application can also show results. Your application can provide a print preview dialog box, showing which colors in a printed image, for example, are out of gamut for the image as it appears on the screen.

For an image that your application prepares, for example, your application can present a print preview dialog box that signifies those colors within the image that the printer cannot accurately reproduce. Your application can also allow users to choose whether and how to match colors in the image with those available on the printer.

You can provide a gamut-checking feature that marks the areas of a displayed image, showing the colors that do not fall within the destination device's gamut. For example, your application can color check an image against a destination device and create a black-and-white version of the image drawn to the display using black to indicate the portions of the source image that are out of gamut. The CSDemo sample application takes this approach. For information on how to obtain the CSDemo application, see Extracting Profiles Embedded in Pictures (page 73).

Creating and Using Device Link Profiles

To accommodate users who use a specific configuration requiring a combination of device profiles and possibly non-device profiles repeatedly over time, your application can create device link profiles. A device link profile offers a means of saving and storing a series of profiles corresponding to a specific configuration in a concatenated format. This feature provides an economy of effort for both your application and its user.

There are many uses for device link profiles. For example, a user might want to store multiple profiles, such as various device profiles and color space profiles associated with the creation and editing of an image.

Most users use the same device configuration to scan, view, and print graphics over a period of time, often soft proofing images before they print them. To enhance your application's soft-proofing feature, you can allow users to store the contents of the profiles involved in the soft-proofing process in a device link profile. Your application can use the appropriate device link profile each time a user enacts the soft-proofing feature, instead of opening a profile reference to each of the profiles to create a color world to pass to the color-matching functions. For additional information about soft proofing, see Providing Soft Proofs (page 93).

A device link profile is especially useful when a scanner application does not embed the source profile in the document containing the image it creates. By storing the scanner's profile, your application eliminates the need to query the user for the appropriate source profile each time the user wants to soft proof using the configuration involving that scanner.

A user may want to see how a scanned image will look when printed using a specific printer. The user may want to look at many images captured on the same scanner at different times before printing the image. Because the same devices are involved in the process, if your application has offered the user the opportunity to create device link profiles, your application could display a list of device link profiles that the user had previously created for various configurations and allow the user to select the appropriate one for the current soft proofing.

Here are the steps your application should take in creating a device link profile:

1. Open the profiles corresponding to the devices and transformations involved in the configuration and obtain references to them.

   To create a device link profile, your application must first obtain references to the profiles involved in the configuration. If the profile for an input device, such as a scanner, is embedded in the document containing the image, you must first extract the profile. For a description of how to obtain a profile reference, see Obtaining Profile References (page 58). For information describing how to extract a profile from a document, see Extracting Profiles Embedded in Pictures (page 73).

2. Create an array containing references to the profiles, specifying the profile references in processing order.

   You supply the profile references as an array of type `CMProfileRef` within a data structure of type `CMConcatProfileSet`. The order of the profiles must correspond to the order in which you want the colors of the image to be processed. For example, for soft proofing an image, you should specify the scanner profile reference first, followed by the printer profile reference, and then the display profile reference because the goal is to match the colors of the scanned image to the color gamut of the printer for which the image is destined and then display the results to the user.

   In the `count` field, specify a one-based number identifying how many profiles the array holds. A device link profile represents a one-way link between devices.

   Here is the `CMConcatProfileSet` data type:`struct CMConcatProfileSet { unsigned shortkeyIndex;/* zero-based */ unsigned shortcount;/* one-based */ CMProfileRefprofileSet[1]; };`

   You must adhere to the rules that govern the type of profiles you can specify in the array. For example, the first and last profiles must be device profiles. For a list of these rules, see `CMConcatProfileSet`.

3. Specify the index corresponding to the profile whose specified CMM is used to perform the processing.

The header of each profile specifies a CMM for that profile. Only one CMM is used for all transformations across the profiles of a device link profile. You identify the profile whose CMM is used by supplying the zero-based index of that profile in the `keyIndex` field of the `CMConcatProfileSet` data type.

> **Important:** See How the ColorSync Manager Selects a CMM (page 51) for a complete description of the ColorSync algorithm for selecting a CMM.

4. Using the CMProfileLocation data type, provide a file specification for the new device link profile.

   If the function `CWNewLinkProfile` is successful, the ColorSync Manager creates a device link profile in the location that you specify, opens a reference to the profile, and returns the profile reference to your application. To tell the ColorSync Manager where to create the new profile, your application must provide a file specification. The ColorSync Manager defines a data structure of type `CMProfileLocation` containing a `CMProfLoc` union that you use to give a file specification.See Listing 4-2 (page 59), which assigns values to a `CMProfileLocation` data structure.

5. Call the CWNewLinkProfile function to create the device link profile.

   After you set up `CMConcatProfileSet` and `CMProfileLocation`, your application can call the function `CWNewLinkProfile`, passing these values to it. If the function completes successfully, it returns a reference to the newly created device link profile.

   Note that you should not embed a device link profile into a document along with an image that uses it, as embedded profiles specify source device characteristics only.

6. Using the CWConcatColorWorld function, create a color world based on the device link profile.

   You can use a device link profile with the general purpose ColorSync Manager functions only. To use a device link profile for a color-matching or color gamut-checking function, you must first create a color world using the `CWConcatColorWorld` function, passing to it a data structure of type `CMConcatProfileSet`. The `CMConcatProfileSet` data structure is the same data type that you used to specify the array of profiles when you created the new device link profile. To create the color world, however, you specify the device link profile as the only member of the `CMConcatProfileSet` array. If the `CWConcatColorWorld` function is successful, it returns a reference to a color world that your application can pass to other general purpose functions for color-matching and color gamut-checking sessions. A device link profile remains intact and available for use again after your application calls the `CWDisposeColorWorld` function to dispose of the concatenated color world.

Considerations

Here are some points to consider about how the ColorSync Manager uses information contained in the profiles comprising a device link profile:

- When you use a device link profile, 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.

- The ColorSync Manager uses the rendering intent specified by the first profile to color match to the second profile, the rendering intent specified by the second profile to color match to the third profile, and so on through the series of concatenated profiles.

When your application is finished with the device link profile, it must close the profile with the `CMCloseProfile`function.

Providing Soft Proofs

Your application can use ColorSync to provide soft-proofing. Soft-proofing enables a user to preview the printed results of a color image on the system's display or local printer without actually outputting the image to the printer that will produce the final image. The destination printer's profile provides the ColorSync Manager with the information required to determine how the colors of the image will appear when printed. You can soft proof an image by showing on the system's display the outcome a printer would produce because most displays support a wider color gamut than do printers. Therefore, a display will probably be able to show all the colors a printer could support.

Providing a feature that simulates the printed outcome for the user to preview can save users considerable time and cost by allowing them to intervene and adjust colors before sending the image to a printing shop. For example, without the ability to soft proof and correct the colors of an image using a color management system such as ColorSync, a graphics designer producing a poster to be printed by a printing press would require the services of a prepress shop to achieve the correct results before sending the image to the printing press. The graphics designer might print the image to a local desktop printer with a color gamut more limited than that of a printing press and then submit the output to the prepress to correct the colors, repeating this process until the results were satisfactory. Your application can eliminate the need for the intermediate steps by allowing the user to color match the image to the color gamut of the final printing press, display the image, and adjust the colors accordingly.

You can use the general purpose color-matching functions `CWMatchPixMap` and `CWMatchBitmap` to perform the color matching, or you can match a list of colors using the `CWMatchColors` function. To use these functions, your application must first define a color world that encompasses the profiles for the devices involved in the soft-proofing process.

For example, suppose a user intends to create a color image by drawing to the display, then color matching the image to the color gamut of the printing press and printing the image to a local desktop printer before delivering it to the printing press. The user intends to repeat this process until he or she is satisfied with the color rendering. To allow the user to do this, your application must build a color world using the profile for the display device, the profile for the printing press, and the profile for the local desktop printer; you must specify the profiles in processing order. Because the process involves three profiles, your application must use the function `CWConcatColorWorld` to set up the color world. Creating a Color World to Use With the General Purpose Functions (page 64) describes how to set up a color world.

You can preserve the series of profiles from a soft-proofing process for future use by creating a device link profile representing the configuration and passing the device link profile to the `CWConcatColorWorld` function to set up a color world. For information on how to create and use a device link profile to build a color world, see Creating and Using Device Link Profiles (page 90).

Your application can also use the QuickDraw-specific `NCMBeginMatching` and `CMEndMatching` functions for soft proofing of a color image drawn to the display that a user wants to color match to the gamut of a printing press and print to a desktop printer.

The `NCMBeginMatching` function matches the colors using the two profiles that you specify, and the `CMEndMatching` function terminates the color-matching session. Because the `NCMBeginMatching` function takes two profiles only—a source profile and a destination profile—you must call sets of these functions to enact soft proofing.

QuickDraw matches to the most recently added profiles first. Therefore, to use the `NCMBeginMatching` and `CMEndMatching` pair to perform soft proofing from a displayed image to a printing press output image to a desktop printer image, you would first call the `NCMBeginMatching` function with the printing press to desktop printer profile references and then call `NCMBeginMatching` with the display to printing press profile references. QuickDraw will color match all drawing from display to printing press and then to the desktop printer.

To use the `NCMBeginMatching` function, you specify the source and destination profiles. Passing `NULL` as the source profile assures that the ColorSync Manager uses the system profile as the source profile. Similarly, passing `NULL` as the destination profile uses the system profile as the destination profile.

Calibrating a Device

A calibration application either creates a profile or tunes a profile to represent the current state of the device.

A profile contains two types of device information: the actual calibration information describing how to perform the color match and the device settings at the time the match was made, for example, paper type, ink flow, or film exposure time. A device may have several profiles, each for a different setting, such as paper type or ink.

Your calibration program should first turn off matching on the device and generate its image. You should then perform the calibration and generate a profile. For related information, see Monitor Calibration and Profiles (page 41)

Accessing a Resource-Based Profile With a Procedure

The ColorSync Manager provides for multiple concurrent accesses to a single profile through the use of a private data structure called a profile reference. When you call the `CMOpenProfile` function to open a profile or the `CMNewProfile`, `CWNewLinkProfile`, or `CMCopyProfile` functions to create or copy a profile, you pass a profile location and the function returns a profile reference. To specify the profile location, you use a structure of type `CMProfileLocation`, as described in Opening a Profile and Obtaining a Reference to It (page 58).

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 `CMProfileLocation` 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 a pointer that may point to data associated with your procedure.

The sample code in Listing 4-18 (page 96) to Listing 4-29 (page 106) demonstrates how to use a profile access procedure to provide access to a resource-based profile.

Note: While the following sample code includes some error handling, more complete error handling is left as an exercise for the reader.

Defining a Data Structure for a Resource-Based Profile

The sample code listings that follow use the application-defined MyResourceLocRec data structure. It stores information to describe a resource-based profile, including

- the resource file specification
- the resource type
- the resource ID
- the resource file reference
- the resource handle
- the profile access procedure pointer
- the resource name

```
struct MyResourceLocRec {
    FSSpec              resFileSpec;
    ResType             resType;
    short               resID;
    short               resFileRef;
    Handle              resHandle;
    CMProfileAccessUPP  proc;
    Str255              resName;
};

typedef struct MyResourceLocRec MyResourceLocRec, *MyResourceLocPtr;
```

The ColorSync Manager defines the CMProfileAccessUPP type as follows:

```
typedef UniversalProcPtr CMProfileAccessUPP;
```

Setting Up a Location Structure for Procedure Access to a Resource-Based Profile

The MyCreateProcedureProfileAccess routine shown in Listing 4-18 (page 96) sets up a `CMProfileLocation` structure for procedure access to a resource-based profile. The MyDisposeProcedureProfileAccess routine, shown in Listing 4-19 (page 97), disposes of memory allocated by MyCreateProcedureProfileAccess. Your application uses these routines (or similar ones that you write) in the following way:

1. Before calling a ColorSync Manager routine such as `CMCopyProfile`, you call the MyCreateProcedureProfileAccess routine to set up a CMProfileLocation structure that you can pass to the ColorSync Manager routine. The location structure specifies your profile-access procedure and may provide other information as well. A sample profile-access procedure is shown in Listing 4-20 (page 98).

2. During the course of its operations, the ColorSync Manager may call your profile-access procedure many times.

3. After the ColorSync Manager routine has completed its operation, and if your application does not need to use the CMProfileLocation structure for another operation, you call the MyDisposeProcedureProfileAccess routine to dispose of memory allocated by MyCreateProcedureProfileAccess.

For the sample MyCreateProcedureProfileAccess routine shown in Listing 4-18 (page 96), you pass a pointer to a CMProfileLocation structure to fill in, a pointer to a file specification for the resource file containing the profile resource, the type of the resource, the ID for the resource, and optionally the name of the resource (stored as a Pascal string, where the first byte is a length byte for the string).

Note: Listing 4-18 (page 96) assumes the profile access routine, MyCMProfileAccessProc, is within the scope of the MyCreateProcedureProfileAccess routine. Optionally, you could add a parameter to pass in a procedure pointer for the profile access routine.

Listing 4-18 Setting up a location structure for procedure access to a resource-based profile

```
OSErr MyCreateProcedureProfileAccess (
                                    CMProfileLocation *profileLocation,
                                    FSSpec *resourceSpec,
                                    Str255 resourceName,
                                    OSType resourceType,
                                    short resourceID)
{
    OSErr               theErr = noErr;
    MyResourceLocPtr    resourceInfo;

    /* Allocate memory for our private resource info structure. */
    resourceInfo = (MyResourceLocPtr) NewPtrClear(sizeof(MyResourceLocRec));
    if (!resourceInfo)
        theErr = MemError();

    if (!theErr)
    {
        /* Set up our private resource info structure. */
        resourceInfo->resFileSpec = *resourceSpec;
        resourceInfo->resType = resourceType;
        resourceInfo->resID = resourceID;
        resourceInfo->resFileRef = 0;
        resourceInfo->resHandle = 0;
        resourceInfo->proc = NewCMProfileAccessProc(MyCMProfileAccessProc);
        /* If a resource name was passed in, copy it to the structure;
            since it's a Pascal string, first byte is length;
            note that BlockMoveData is faster than BlockMove for a
            move that involves data only. */
        if (resourceName)
            BlockMoveData(resourceName, resourceInfo->resName,
                                resourceName[0]+1);
        /* set up the profile location structure */
        profileLocation->locType = cmProcedureBasedProfile;
        profileLocation->u.procLoc.refCon = (void*) resourceInfo;
        profileLocation->u.procLoc.proc = resourceInfo->proc;
    }
    return theErr;
}
```

If the MyCreateProcedureProfileAccess routine is able to set up the profile location pointer for procedure access to a resource-based profile, it returns a value of `noErr`.

Disposing of a Resource-Based Profile Access Structure

Your application calls the MyDisposeProcedureProfileAccess routine (Listing 4-19 (page 97)) to dispose of any memory allocated by the MyCreateProcedureProfileAccess routine (Listing 4-18 (page 96)).

Listing 4-19 Disposing of a resource-based profile access structure

```
void MyDisposeProcedureProfileAccess (CMProfileLocation *profileLocation)
{
    DisposeRoutineDescriptor(profileLocation->u.procLoc.proc);

    /* Dispose of our private resource info structure. */
    DisposePtr((Ptr)profileLocation->u.procLoc.refCon);
}
```

This routine first disposes of the universal procedure pointer to your profile access procedure, then disposes of the pointer used to store resource data in a MyResourceLocRec structure.

Responding to a Procedure-Based Profile Command

For information on the procedure declaration for a profile access procedure, see `MyCMProfileAccessProc`. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed, passing a command constant that specifies the current command. Your profile access procedure must be able to respond to each of the following command constants, which are described in *ColorSync Manager Reference*:

```
enum {
    cmOpenReadAccess    = 1,
    cmOpenWriteAccess   = 2,
    cmReadAccess        = 3,
    cmWriteAccess       = 4,
    cmCloseAccess       = 5,
    cmCreateNewAccess   = 6,
    cmAbortWriteAccess  = 7,
    cmBeginAccess       = 8,
    cmEndAccess         = 9
};
```

The profile access procedure shown in Listing 4-20 (page 98), `MyCMProfileAccessProc`, consists of a single switch statement, which calls the appropriate routine based on the value of the `command` parameter. Each of the nine routines called by `MyCMProfileAccessProc` is described and listed in the sections that follow Listing 4-20 (page 98), and each refers back to Listing 4-20 (page 98).

Listing 4-20 Responding to a procedure-based profile command

```
pascal OSErr MyCMProfileAccessProc (long command,
                                    long offset,
                                    long *sizePtr,
                                    void *dataPtr,
                                    void *refConPtr)
{
    OSErr   theErr = noErr;
    switch (command)
    {
        case cmBeginAccess:
            theErr = DoBeginAccess(refConPtr);
            break;

        case cmCreateNewAccess:
            theErr = DoCreateNewAccess(refConPtr);
            break;

        case cmOpenReadAccess:
            theErr = DoOpenReadAccess(refConPtr);
            break;

        case cmOpenWriteAccess:
            theErr = DoOpenWriteAccess(sizePtr, refConPtr);
            break;

        case cmReadAccess:
            theErr = DoReadAccess(offset, sizePtr, dataPtr, refConPtr);
            break;

        case cmWriteAccess:
            theErr = DoWriteAccess(offset, sizePtr, dataPtr, refConPtr);
            break;

        case cmCloseAccess:
            theErr = DoCloseAccess(refConPtr);
            break;

        case cmAbortWriteAccess:
            theErr = DoAbortWriteAccess(refConPtr);
            break;

        case cmEndAccess:
            theErr = DoEndAccess(refConPtr);
            break;

        default:
            theErr = paramErr;
            break;
    }

    return theErr;
}
```

Note that the MyCMProfileAccessProc routine passes its parameter data as necessary to the routines it calls. The parameters have the following values:

command
: A command value indicating the operation to perform. The possible values for command constants are shown elsewhere in this section.

offset
: For read and write operations, the offset from the beginning of the profile at which to read or write data.

size
: 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 output 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 output, for a read operation, contains the data that was read.

refConPtr
: A reference constant pointer that can store private data for the MyCMProfileAccessProc procedure. For example, Listing 4-18 (page 96) shows how to set up a location structure for procedure access to a resource-based profile. That routine sets the location structure's refCon field to a pointer to a MyResourceLocRec structure, which is described in Defining a Data Structure for a Resource-Based Profile (page 95). That same structure pointer is passed to the MyCMProfileAccessProc routine in the refConPtr parameter, and provides access to all the stored information about the resource location.

Handling the Begin Access Command

When your application calls the `CMOpenProfile` routine, specifying as a location a procedure-based profile, the ColorSync Manager invokes your specified profile access procedure with the cmBeginAccess command. This gives your procedure an opportunity to perform any required initialization or validation tasks, such as determining whether the data pointed to by the `refcon` parameter is valid. If your procedure returns an error (any value except `noErr`), the ColorSync Manager will not call your profile access procedure again.

For the cmBeginAccess command, the sample profile access procedure shown in Listing 4-20 (page 98) calls the DoBeginAccess routine, shown in Listing 4-21 (page 99). DoBeginAccess interprets the `refcon` parameter as a MyResourceLocPtr type. If the parameter does not have a resource type of kProcResourceType, DoBeginAccess returns an invalid profile error, which effectively cancels the procedure-based profile access.

Listing 4-21 Handling the begin access command

```
static OSErr DoBeginAccess (void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;

    resourceInfo->resFileRef = 0;

    if (resourceInfo->resType != kProcResourceType)
        theErr = cmInvalidProfileLocation;
    else
        theErr = noErr;

    return theErr;
}
```

Handling the Create New Access Command

When your application calls the `CMCopyProfile` or `CMUpdateProfile` routine, specifying as a location a procedure-based profile, the ColorSync Manager invokes the specified profile access procedure with the cmBeginAccess command, as described in Handling the Begin Access Command (page 99).

If your profile access procedure returns without error, ColorSync calls the procedure again with the cmCreateNewAccess command. Your procedure should create a new data stream for the actual physical location of the profile. The size of the profile is not known at this point.

For the cmCreateNewAccess command, the sample profile access procedure shown in Listing 4-20 (page 98) calls the DoCreateNewAccess routine. DoCreateNewAccess interprets the `refcon` parameter as a MyResourceLocPtr type, and calls the Toolbox routine FSpCreateResFile to create an empty resource fork based on the file specification provided by the MyResourceLocPtr type. If the resource fork does not already exist and cannot be created, DoCreateNewAccess returns an error.

Note that for this example, the file type for a resource-based profile was chosen arbitrarily to be `'rprf'`.

Listing 4-22 Handling the create new access command

```
OSErr DoCreateNewAccess (void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;

    FSpCreateResFile(&(resourceInfo->resFileSpec), '????', 'rprf', 0);
    theErr = ResError();
    if (theErr == dupFNErr)
        theErr = noErr;

    return theErr;
}
```

Handling the Open Read Access Command

When your application calls a ColorSync Manager routine to read information from a procedure-based profile, the ColorSync Manager first calls your profile access procedure with the cmOpenReadAccess command. Then it calls your profile access routine once for each read session. The sample profile access procedure shown in Listing 4-20 (page 98) calls the DoOpenReadAccess routine.

The DoOpenReadAccess routine shown in Listing 4-23 (page 101) uses information from the refcon parameter, interpreted as type MyResourceLocPtr, to open the resource fork for the resource-based profile with read permission. If it can open the resource file, DoOpenReadAccess then attempts to load the profile resource.

The DoOpenReadAccess routine shows good citizenship by saving the current resource file before performing its operations and restoring the resource file afterward.

Listing 4-23 Handling the open read access command

```
static OSErr DoOpenReadAccess (void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;
    short               currentResFile;

    /* Save current resource file. */
    currentResFile = CurResFile();

    /* Open the file's resource fork. */
    resourceInfo->resFileRef = FSpOpenResFile(&(resourceInfo->resFileSpec),
fsRdPerm);
    theErr = ResError();

    /* Get the resource handle, but don’t force it to be loaded into memory. */
    if (!theErr)
    {
        SetResLoad(false);
        resourceInfo->resHandle = GetResource(resourceInfo->resType,
                                            resourceInfo->resID);
        theErr = ResError();
        SetResLoad(true);
    }

    /* Restore previous resource file. */
    UseResFile(currentResFile);

    return theErr;
}
```

Handling the Open Write Access Command

When your application calls the `CMUpdateProfile` routine to update a procedure-based profile or the `CMCopyProfile` routine to copy a profile, the ColorSync Manager calls your profile access procedure with the cmOpenWriteAccess command. The sample profile access procedure shown in Listing 4-23 (page 101) calls the DoOpenWriteAccess routine.

The DoOpenWriteAccess routine shown in Listing 4-24 (page 102) uses information from the refcon parameter, interpreted as type MyResourceLocPtr, to open the resource fork for the resource-based profile with read/write permission. If it can open the resource file, DoOpenWriteAccess then attempts to open the specified profile resource. If it can’t open the resource, DoOpenWriteAccess creates a new resource. It then sets the size of the resource based on the passed setProfileSize pointer value and updates the resource file.

The DoOpenWriteAccess routine shows good citizenship by saving the current resource file before performing its operations and restoring the resource file afterward.

Note: If the cmOpenWriteAccess command succeeds, the ColorSync Manager guarantees an eventual call to the profile access procedure with the cmCloseAccess command, possibly after multiple cmWriteAccess commands, and possibly after a cmAbortWriteAccess command.

Listing 4-24 Handling the open write access command

```
static OSErr DoOpenWriteAccess (long *setProfileSize, void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;
    Size                resourceSize;
    short               currentResFile;

    /* Save current resource file. */
    currentResFile = CurResFile();

    /* Open the file's resource fork. */
    resourceInfo->resFileRef = FSpOpenResFile(&(resourceInfo->resFileSpec),
                                              fsRdWrPerm);
    theErr = ResError();

    /* Get the resource handle, but don’t force it to be loaded into memory. */
    if (!theErr)
    {
        SetResLoad(false);
        resourceInfo->resHandle = GetResource(resourceInfo->resType,
                                              resourceInfo->resID);
        theErr = ResError();
        SetResLoad(true);
    }

    /* Call GetResourceSizeOnDisk to see if resource is already there. */
    if (!theErr)
    {
        /* Get size of the resource. */
        resourceSize = GetResourceSizeOnDisk(resourceInfo->resHandle);
        theErr = ResError();
    }

    /* If the above call to GetResourceSizeOnDisk returns resNotFound,
        then we need to create a new resource */
    if (theErr == resNotFound)
    {
        /* Allocate a temporary handle just so that we can call AddResource. */
        resourceInfo->resHandle = NewHandle(sizeof(long));
        theErr = MemError();

        /* Add resource to the file and release the temp handle. */
        if (!theErr)
        {
            AddResource(resourceInfo->resHandle, resourceInfo->resType,
                        resourceInfo->resID, resourceInfo->resName);
            theErr = ResError();
            ReleaseResource(resourceInfo->resHandle);
        }

    /* Get the resource handle, but don’t force it to be loaded into memory. */
```

```
        if (!theErr)
        {
            SetResLoad(false);
            resourceInfo->resHandle = GetResource(resourceInfo->resType,
                                                  resourceInfo->resID);
            theErr = ResError();
            SetResLoad(true);
        }
    }

    /* Change the resource size to fit the profile. */
    if (!theErr)
    {
        SetResourceSize(resourceInfo->resHandle, *setProfileSize);
        theErr = ResError();
    }

    /* Force an update of the resource file. */
    if (!theErr)
    {
        UpdateResFile(resourceInfo->resFileRef);
        theErr = ResError();
    }

    /* Restore previous resource file. */
    UseResFile(currentResFile);

    return theErr;
}
```

Handling the Read Access Command

When your application calls a ColorSync Manager routine to read information from a procedure-based profile, the ColorSync Manager first calls your profile access procedure with the cmOpenReadAccess command, as described in Handling the Open Read Access Command (page 100). Your profile access routine can be called with the cmReadAccess command at any time after the cmOpenReadAccess command is called. When the sample profile access procedure shown in Listing 4-20 (page 98) receives the cmReadAccess command, it calls the DoReadAccess routine.

The DoReadAccess routine shown in Listing 4-25 (page 103) uses the refcon parameter, interpreted as type MyResourceLocPtr, to get a resource handle for the resource-based profile. From other parameters, it gets values for the offset at which to start reading, the number of bytes to read, and a pointer to a buffer in which to store the data that it reads. It then calls the Toolbox routine ReadPartialResource to do the actual reading.

If an error occurs while reading, DoReadAccess returns the error.

Listing 4-25 Handling the read access command

```
static OSErr DoReadAccess ( long offset,
                            long *sizePtr,
                            void *dataPtr,
                            void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;

    ReadPartialResource(resourceInfo->resHandle,
                            offset, dataPtr, *sizePtr);
    theErr = ResError();

    return theErr;
}
```

Handling the Write Access Command

When your application calls the `CMUpdateProfile` routine to update a procedure-based profile, the ColorSync Manager first calls your profile access procedure with the cmOpenWriteAccess command. The DoOpenWriteAccess routine shown in Listing 4-24 (page 102) performs certain operations to prepare to write a resource-based profile.

Your profile access routine can be called with the cmWriteAccess command at any time after the cmOpenWriteAccess command is called. When the sample profile access procedure shown in Listing 4-20 (page 98) receives the cmWriteAccess command, it calls the DoWriteAccess routine.

The DoWriteAccess routine shown in Listing 4-26 (page 104) uses the refcon parameter, interpreted as type MyResourceLocPtr, to get a resource handle for the resource-based profile. From other parameters, it gets values for the offset at which to start writing, the number of bytes to write, and a pointer to a buffer from which to get the data that it writes. It then calls the Toolbox routine WritePartialResource to do the actual writing.

If an error occurs while writing, DoWriteAccess returns the error.

Note: After ColorSync calls the profile access procedure with the cmWriteAccess command, ColorSync is guaranteed to eventually call the profile access procedure with the cmCloseAccess command—possibly after additional calls with the cmWriteAccess command, and possibly after a call with the cmAbortWriteAccess command.

Listing 4-26 Handling the write access command

```
static OSErr DoWriteAccess (long offset,
                            long *sizePtr,
                            void *dataPtr,
                            void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;

    WritePartialResource(resourceInfo->resHandle,
                            offset, dataPtr, *sizePtr);
    theErr = ResError();

    return theErr;
}
```

Handling the Close Access Command

The ColorSync Manager calls your profile access procedure with the cmCloseAccess command to indicate that reading or writing is finished for the moment. A cmCloseAccess command can be followed by a cmOpenReadAccess command to begin reading again, a cmOpenWriteAccess command to begin writing again, or a cmEndAccess command to terminate the procedure-based profile access.

The sample profile access procedure shown in Listing 4-20 (page 98) calls the DoCloseAccess routine.

The DoCloseAccess routine shown in Listing 4-27 (page 105) uses information from the refcon parameter, interpreted as type MyResourceLocPtr, to close and update the resource file for the resource-based profile. If DoCloseAccess is unsuccessful, it returns an error value.

Listing 4-27 Handling the close access command

```
static OSErr DoCloseAccess (void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;

    /* Close and update resource file. */
    if (resourceInfo->resFileRef)
    {
        CloseResFile(resourceInfo->resFileRef);
        theErr = ResError();
        resourceInfo->resFileRef = 0;
    }
    else theErr = paramErr;

    return theErr;
}
```

Handling the Abort Write Access Command

If an error occurs between a `cmOpenWriteAccess` command and a `cmCloseAccess` command, the ColorSync Manager calls your profile access procedure with the `cmAbortWriteAccess` command. This allows your access procedure to perform any cleanup necessary for the partially written profile.

For the `cmAbortWriteAccess` command, the sample profile access procedure shown in Listing 4-20 (page 98) calls the DoAbortWriteAccess routine.

The DoAbortWriteAccess routine shown in Listing 4-28 (page 106) uses information from the refcon parameter, interpreted as type MyResourceLocPtr, to call the Toolbox routine RemoveResource to delete the partially written resource. If DoAbortWriteAccess is unsuccessful, it returns an error value.

Note: The ColorSync Manager will call your profile access procedure with the `cmCloseAccess` command after a `cmAbortWriteAccess` command.

Listing 4-28 Handling the abort write access command

```
static OSErr DoAbortWriteAccess (void *refcon)
{
    OSErr               theErr;
    MyResourceLocPtr    resourceInfo = refcon;

    /* Delete the resource that we started. */
    if (resourceInfo->resHandle)
    {
        RemoveResource(resourceInfo->resHandle);
        theErr = ResError();
    }
    else theErr = paramErr;

    return theErr;
}
```

Handling the End Access Command

When access to a procedure-based profile is complete, the ColorSync Manager calls your profile access procedure with the `cmEndAccess` command. This allows your procedure to do any final cleanup, such as freeing memory allocated by the procedure.

For the `cmEndAccess` command, the sample profile access procedure shown in Listing 4-20 (page 98) calls the DoEndAccess routine. Because there is no additional memory to free or other cleanup to take care of, the DoEndAccess routine shown in Listing 4-29 (page 106) does nothing.

Note: The MyCreateProcedureProfileAccess routine, shown in Listing 4-18 (page 96), does allocate memory, which is freed by a call to the MyDisposeProcedureProfileAccess routine, shown in Listing 4-19 (page 97). Your application calls the MyCreateProcedureProfileAccess routine before calling a ColorSync Manager routine such as `CMCopyProfile` with a procedure-based profile. After the copy is complete, your application calls the MyDisposeProcedureProfileAccess routine to perform any necessary deallocation.

Listing 4-29 Handling the end access command

```
pascal OSErr DoEndAccess (void *refcon)
{
    OSErr   theErr = noErr;

    return theErr;
}
```

Summary of the ColorSync Manager

This section provides a quick-reference summary of the functions, data types, and constants that make up the ColorSync Manager programming interface.

Developing ColorSync-Supportive Device Drivers

This section describes how you can use the ColorSync Manager to provide ColorSync-supportive device drivers for peripherals. It first describes how input, display, and output devices work with color profiles. It then discusses how devices interact with color management modules (CMMs). Next it describes what you must do to provide minimum ColorSync support, as well as how you can provide more extensive support. Finally, it uses a QuickDraw-based printer device driver to demonstrate some of the color-matching features a device driver can provide.

Read this section if your device driver for an input, display, or output device will support the ColorSync Manager and allow users to produce color-matched images.

Before you read this section, you should read Overview of ColorSync (page 27) and Overview of Color and Color Management Systems (page 13). These sections provide an overview of ColorSync, explain color theory and color management systems, and define key terms.

Although the features described here are commonly provided by printer device drivers, the code samples in Developing ColorSync-Supportive Applications (page 49) may also be of use in developing your device driver.

ColorSync Manager Reference describes constants, data structures, functions, and result codes for ColorSync-supportive applications and device drivers.

ColorSync Version Information (page 141) describes the `Gestalt` information, shared library version numbers, CMM version numbers, and ColorSync header files you use with different versions of the ColorSync Manager. It also includes CPU and Mac OS system requirements.

About ColorSync-Supportive Device Driver Development

A device driver that supports ColorSync should provide at least one profile for the device. In addition, it can provide its own CMM, designed to perform the best possible color matching for the device. If you are creating your own CMM, you should read Developing Color Management Modules (page 121) and *ColorSync Manager Reference*.

Devices and Their Profiles

To assess the way each device interprets color, color scientists and profile developers perform device characterizations. This process, which entails measuring the gamut of a device, yields a color profile for that device. For an overview of profiles, see Profiles (page 30).

Device profiles are of paramount importance to any color management system because they characterize the unique color behavior of each device and provide the data needed for color matching and color conversion. Device profiles are used by CMMs that perform the low-level calculations required to match colors from a source device to a destination device.

The ICC defines a device profile class for each of three types of devices:

- An input device, such as a scanner or a digital camera.
- A display device, such as a monitor or a liquid crystal display.
- An output device, such as a printer.

These classes are described in detail in Profile Classes (page 31).

Each device profile class has its own signature. The ColorSync constants for these signatures are described in *ColorSync Manager Reference*. You can create a device driver for any of the device classes. When you create a profile for your device, you specify the signature in the profile header's `profileClass` field. For more information on profile headers, see *ColorSync Manager Reference*

Whether you create a profile for your device or obtain one from a profile vendor, your device driver must provide at least one profile for its device. However, you can provide more than one profile for the same device to characterize different states. Although a printer that your device driver supports may have a number of profiles for different conditions, such as the use of foils or different grades of paper, all of its profiles will use the same profile signature, `cmOutputClass`.

The Profile Format and Its Cross-Platform Use

Device profiles follow the ICC profile format, an industry standard described in Profiles (page 30). You can provide a single profile or a set of profiles that can be used across different operating systems for the device your driver supports. The common profile format specified by the ICC allows end users to transparently move profiles and images with embedded profiles between different operating systems.

Note: The ICC publishes the International Color Consortium Profile Format Specification.To obtain a copy of the specification, visit the ICC Web site at <http://www.color.org/>.

The profile structure is defined as a header followed by a tag table which, in turn, is followed by a series of tagged elements that your device driver can access randomly and individually. Using ColorSync Manager functions, you can read the profile header and modify its contents and you can get and set individual tags and their element data.

ColorSync Profile Format Version Numbers

This document uses 2.x to refer to ColorSync profiles for ColorSync Manager version 2.0 and greater, as described in ColorSync and ICC Profile Format Version Numbers (page 30). Version 2.x profiles require more information and are larger than ColorSync 1.0 profiles, which were originally memory based. Because version 2.x profiles are larger, they are disk-based. The ICC profile format specification defines how profiles can be stored as disk files and how profiles can be embedded in common graphics file formats such as PICT and TIFF. The ColorSync Manager provides the `CMProfileLocation` data structure to identify the location of a profile. It also provides functions you can use to embed a profile in or extract if from a PICT file, as described in *ColorSync Manager Reference*.

Storing and Handling Device Profiles

Device profiles reside in the ColorSync Profiles folder, in pictures, or with device drivers. Files that contain profiles store profile data in the data fork and have a file type of `'prof'`.

By convention, profiles not embedded in image documents are stored in the ColorSync Profiles folder, as described in Profile Search Locations (page 34). You can store your profile files wherever you want, but if you want other drivers or applications to have access to them, you should store them in the ColorSync Profiles folder. Applications that perform soft proofing or gamut checking can use ColorSync Manager routines to search the folder for specific types of profiles to provide a pop-up menu or list to the user. If your profiles are not available, these applications will not be able to include these profiles in menus or lists, and will not be able to color match to your device (unless they provide a profile for your device themselves).

Some applications may place special-purpose profiles for your device in the ColorSync Profiles folder. For this reason, when your device driver itself displays a pop-up menu or list to the user, you should search not only your private profile location storage, if you use one, but also the ColorSync Profiles folder, to make sure that you offer a complete list of available profiles for your device.

The ColorSync Profiles folder can contain both ColorSync 1.0 profiles and version 2.x profiles. However, your device driver will be able to search for only version 2.x profiles. This is because ColorSync Manager 2.x search functions do not acknowledge ColorSync 1.0 profiles. Support for 1.0 profiles may be even more limited in the future. For more information on this and other limitations of the 1.0 profile format, see ColorSync 1.0 Profiles and Version 2.x Profiles (page 145).

How a Device Driver Uses Profiles

For most ColorSync Manager functions that your device driver calls, you will need to supply references to profiles for both the source device on which the image was created and the destination device for which it is to be color matched and where it will be rendered.

The driver for an input device such as a scanner typically embeds the scanner profile used to create the image in the document containing the image. The driver for a device that displays an existing image on the system's display or a printer device that prints a color-matched image typically extracts the embedded profile that accompanies the image from the document containing the image, and uses that profile as the source profile when matching.

Images created using input devices are commonly color matched using the profile for the input device as the source profile and the system profile for the display as the destination profile. Setting Default Profiles (page 33) describes how to set the default system profile and other default profiles. Images that are created, depicted, or modified on a display device and that are destined for an output device such as a printer are color matched using the profile for the display as the source profile and the printer's profile as the destination profile.

To use a profile, you must first obtain a reference to the profile. For a description of how to do this, see Obtaining Profile References (page 58).

Devices and Color Management Modules

Your device driver can use the color conversion functions, described in *ColorSync Manager Reference*, to convert colors between color spaces belonging to the same base family without relying on a CMM. However, color matching, gamut checking, providing color rendering dictionaries to PostScript printers, and other tasks you perform using ColorSync Manager functions all require use of a CMM. It is the CMM that actually carries out the work of the ColorSync Manager functions, such as performing the low-level calculations required to match colors from a source device to a destination device.

If your ColorSync-supportive device driver can use the Apple-supplied default CMM, you need only provide one or more profiles for your device. However, you may want to provide a custom CMM that is optimized for your device and its profiles. For example, a profile can provide private tags containing information a custom CMM might use to achieve better results for the device.

To provide your own CMM, you can create one or obtain one from a vendor. For information describing how to create a CMM, see Developing Color Management Modules (page 121) and *ColorSync Manager Reference*

For additional information on CMMs, see Setting a Preferred CMM (page 37) and How the ColorSync Manager Selects a CMM (page 51).

Providing ColorSync-Supportive Device Drivers

Your ColorSync-supportive device driver can provide users with various color-matching features based on the type of device you support. This section describes:

- Providing Minimum ColorSync Support (page 112)
- Providing More Extensive ColorSync Support (page 112)

Providing Minimum ColorSync Support

The minimum level of ColorSync support you should provide differs depending on the type of device your driver supports.

For a scanner, you should embed the scanner profile used to create the image in the document containing the image; this is also referred to as tagging an image. If you do not tag the image with the profile, you should at least make the profile for the image available so that it can be used for color matching. If you do not provide the scanner profile, an application or driver that attempts to color match the scanned image will use the system profile as the source profile and may produce results inconsistent with the colors of the original image.

For a display device driver or a printer device driver, you must preserve images tagged with a profile by not stripping out picture comments used to embed profiles or by leaving profiles in documents that use other methods to include them. For example, if your driver displays or prints PICT files but does not perform color matching, your driver should not strip out the ColorSync-related picture comments that are used to embed profiles in PICT files, begin and end use of a specific profile, and enable and disable color matching. Even though your driver may not make use of the comments, another display or printer driver or an application may use them.

If you don’t perform color matching but you want to allow other applications to produce images that are color matched for your device, you should provide a device profile to be used as the destination profile. If you provide a profile for your display or printer and place it in the ColorSync Profiles folder, applications that perform color matching can use it to create a color-matched image expressed in the colors of your device’s gamut. A user can then print a color-matched image using the printer your driver supports.

Providing More Extensive ColorSync Support

Instead of relying on an application to color match an image for your printer, your printer driver can color match the image itself before sending it to the printer. To perform color matching, your printer driver must obtain a reference to the source profile. Documents containing images to be printed often contain an embedded profile along with the image. To use the source profile, your printer driver must be able to extract it. If an image is not accompanied by a source profile, the default system profile is used, as described in Setting Default Profiles (page 33). In this case, your driver should provide an interface that allows the user to select the rendering intent to be used. Rendering intents are described in Rendering Intents (page 37).

You can provide an interface that offers additional features. Your interface can

- allow a user to turn ColorSync color matching on or off before printing
- offer pop-up menus, allowing the user to choose
  - the rendering method to be used in color matching the image (perceptual, colorimetric, or saturation)
  - the color-image quality (normal, draft, or best)

Some of these features are discussed below and in Developing ColorSync-Supportive Applications (page 49).

Developing Your ColorSync-Supportive Device Driver

This section describes how your device driver can implement certain color matching and related features with ColorSync. It includes the following:

- Determining If the ColorSync Manager Is Available (page 113)
- Interacting With the User (page 113)
- Color Matching an Image to Be Printed (page 119)

Many of the tasks that your device driver performs to support ColorSync can also be performed by other kinds of color-matching applications. Some of these tasks are mentioned here, but not explained in detail. For a list of code samples shown elsewhere, see Developing Your ColorSync-Supportive Application (page 55).

Determining If the ColorSync Manager Is Available

To determine if the ColorSync Manager (version 2.x) is available, call the `Gestalt` function with the `gestaltColorMatchingVersion` selector. For sample code that demonstrates how to perform this operation, see Determining If the ColorSync Manager Is Available (page 56). ColorSync constants for use with the `Gestalt` function are described in *ColorSync Manager Reference*. For related information on ColorSync versions, see Table 7-1 (page 142), which lists version numbers for releases of the ColorSync Manager, along with corresponding shared library version numbers, `Gestalt` selector codes, and hardware and system requirements.

Interacting With the User

Using lists and dialog boxes, you can provide choices that influence the color-matching process. For example, you can offer any of the following:

- A list of profiles to select from. You can allow the user to choose the appropriate profile for your printer in its current state. To provide a list of profiles for the user to select from, you must first search for the relevant profiles:
  - ❏ Starting with version 2.5 of the ColorSync Manager, you should use the algorithm shown in Performing Optimized Profile Searching (page 81) to search for profiles. That algorithm uses `CMIterateColorSyncFolder`.
  - ❏ For versions of the ColorSync Manager prior to 2.5, you can use the functions described in *ColorSync Manager Reference*.
- A dialog box for specifying how the image will be color matched. If the source profile is embedded with the image, the source profile specifies the rendering intent to be used. However, if the source profile is not provided and the system profile is used as the source profile, you should allow the user to select the rendering intent to be used. After the user chooses a rendering intent, you can use the selection to set the source profile's header. Setting a User-Selected Rendering Intent (page 114) explains this process.
- A dialog box for choosing which color-matching quality of image to produce. A user may want to produce a draft of the image quickly for review before producing the best possible quality of the image. After the user chooses a color-matching quality, you can use the selection to set the source profile's header. Setting a User-Selected Color-Matching Quality Flag (page 116) explains how to do this.
- A dialog box for turning ColorSync color matching on or off before printing. If an application that creates or modifies an image has already performed color matching using your printer profile as the destination profile, the user might want to turn off color matching. To provide this capability, your driver should support the `PrGeneral` function with the `enableColorMatchingOp` operation code. For information on the `PrGeneral` function, see Inside Macintosh: Imaging With QuickDraw. The `enableColorMatchingOp` operation code constant defined by the ColorSync Manager is described in *ColorSync Manager Reference*.

Setting a User-Selected Rendering Intent

The ColorSync Manager supports the four standard rendering intents defined by the ICC—perceptual, relative colorimetric, saturated, and absolute colorimetric. Every profile supports these four intents, which are commonly used to match the colors of a source image to the color gamut of the destination device in the most optimum way for the type of image. These intents are described in detail in Rendering Intents (page 37).

If the source profile is embedded with the image, the source profile specifies the rendering intent to be used. However, if the source profile is not available and the system profile must be used as the source profile, you should allow the user to select the rendering intent to be used.

Note: Starting with ColorSync 2.5, your application can call `CMGetDefaultProfileBySpace` to obtain an appropriate source profile for matching, rather than using the default system profile. However, you may still wish to allow the user to specify a rendering intent.

To allow users to choose the rendering intent most appropriate for color matching a graphical image, you can provide a pop-up menu or a dialog box identifying the rendering intent options available. By providing a description of the available rendering intents, you can help a user select the rendering intent that best maintains important aspects of the image.

Color professionals and technically-sophisticated users are likely to be familiar with the ICC terms for rendering intent and the gamut-matching strategies they represent. If your application is aimed at novice users, however, you may prefer to use a simplified terminology based on the typical image content associated with a rendering intent, as described in Table 3-1 (page 37). For example, you might note the following:

- For perceptual matching, all the colors of a given gamut may be scaled to fit within another gamut. This intent is the best choice for realistic images, such as scanned photographs.
- For saturation matching, the relative saturation of colors is maintained from gamut to gamut. Rendering the image using this intent gives the strongest colors and is the best choice for bar graphs and pie charts, in which the actual color displayed is less important than its vividness.
- For relative colorimetric matching, the colors that fall within the gamuts of both devices are left unchanged. Some colors in both images will be exactly the same, a useful outcome when colors must match quantitatively. This intent is best suited for logos or spot colors.
- For absolute colorimetric matching, a close appearance match may be achieved over most of the tonal range, but if the minimum density of the idealized image is different from that of the output image, the areas of the image that are left blank will be different. Colors that fall within the gamuts of both devices are left unchanged.

After the user selects the intent to be used, you must modify the `renderingIntent` field of the system profile's header to reflect the choice. To put the rendering intent chosen by the user in the profile header, follow these steps:

1. **Obtain a profile reference to the system profile.**

   Identifying the Current System Profile (page 60) describes how to do this.

2. **Get the profile header of the system profile.**

   You call the function `CMGetProfileHeader`, passing the profile reference, to obtain the profile's header. The function returns the profile header using a union of type `CMAppleProfileHeader`. You can use this function for both ColorSync 1.0 profiles and version 2.x profiles.

   For a version 2.x profile, you use the data structure `CM2Header`. For a version 1.0 profile, you use the `CMHeader` data structure. For more information on profile headers, see *ColorSync Manager Reference*.

3. **Assign the new rendering intent to the header field.**

   To assign a rendering intent to the system profile header's `renderingIntent`

   `enum { cmPerceptual= 0, cmRelativeColorimetric= 1, cmSaturation= 2, cmAbsoluteColorimetric= 3 };` field, use the constants defined by the following enumeration:

   These constants are described in *ColorSync Manager Reference*.

4. **Set the modified profile header of the system profile.**

   After you assign the rendering intent, you must replace the header by calling the function `CMSetProfileHeader`. You can use this function to set a header for a version 1.0 or a version 2.x ColorSync profile. You pass the header using the union `CMAppleProfileHeader`.

You can now use the system profile to create a color world for the color-matching process. For information on how to create a color world, see Creating a Color World to Use With the General Purpose Functions (page 64).

Important: When you call `CMSetProfileHeader`, the profile header is modified temporarily. The rendering intent change is discarded when you call the function `CMCloseProfile`. To preserve the change, you must call the function `CMUpdateProfile`.

Listing 5-1 (page 118) includes code that uses the `cmSaturation` constant to set the rendering intent for a profile.

Setting a User-Selected Color-Matching Quality Flag

The ColorSync Manager provides a feature, called the quality flags settings, that controls the quality of the color-matching process in relation to the time required to perform the match. This feature, which is not a standard feature defined by the ICC profile format specification, works by letting you manipulate certain bits of the profile header's `flags` field. There are three quality flag settings: normal, draft, and best. For a description of the profile header's `flags` field, see *ColorSync Manager Reference*.

Normal mode is the default setting. Color matching using draft mode takes the least time and produces the least exact results. Color matching using best mode takes the longest time but produces the finest results.

Users sometimes want to produce review drafts of images quickly before expending the time to produce the best-quality final copy. Your interface can allow them this flexibility by offering a dialog box that provides the three options.

After the user selects the color-matching quality, you can use the selection to set the appropriate bits of the source profile's `flags` field. To set the color-matching quality chosen by the user, follow these steps:

1. Obtain a profile reference to the source profile.

   Obtaining Profile References (page 58) describes how to do this.

2. Get the profile header of the source profile.

   You call the function `CMGetProfileHeader`, passing the profile reference, to obtain the profile's header. The function returns the profile header using a union of type `CMAppleProfileHeader`.

3. Optionally, test the current setting of the source profile header's flags.

   The `flags` field of the source profile header is a long word coded in big-endian notation. Big-endian notation is a means of encoding data in which the first byte within 16-bit and 32-bit quantities is the most significant. The ICC profile consortium reserves the first 2 bits of the low word for its own use. The least significant 2 bits of the high word constitute the quality flag settings used to specify the quality for the color matching. The bit definitions for the `flags` field are shown in *ColorSync Manager Reference*.

   To evaluate and interpret the current setting of the quality flags bits, you can take these steps, in order:

   - Right-shift by 16 bits.
   - Mask off the high 14 bits.
   - Compare the result with values defined by the following enumeration:

```
enum
{
 cmNormalMode = 0,
 cmDraftMode = 1,
 cmBestMode = 2
};
```

These constants are described in *ColorSync Manager Reference*.

4. Set the quality flags bits to the user-selected value.

   To set the quality flag, you can use the constants defined by the enumeration provided by the ColorSync Manager and shown in step 3.

5. Set the source profile with the modified profile header.

   After you set the `flags` field based on the user's selection, you must replace the header by calling the function `CMSetProfileHeader`. You pass the header using the union `CMAppleProfileHeader`.

You can now use the source profile to create a color world for the color-matching process. For information on how to create a color world, see Creating a Color World to Use With the General Purpose Functions (page 64).

Important: When you call `CMSetProfileHeader`, the profile header is modified temporarily. Changes to the `flags` field are discarded when you call the function `CMCloseProfile`. To preserve the change, you must call the function `CMUpdateProfile`.

Listing 5-1 (page 118) shows how to set the system profile's quality flag to best mode for producing the highest-quality color-matched image. It also sets the rendering intent to saturation before setting up a color world based on the modified system profile and the printer profile.

The `MySetHeader` function shown in Listing 5-1 (page 118) initializes the `CMProfileRef` data structures it will use for the system profile and the printer profile before it calls the following two functions—the ColorSync Manager function `CMGetSystemProfile` to obtain a reference to the system profile and its own function `MyGetPrinterProfile` to obtain a reference to the profile for its printer.

The source profile (in this case, the system profile), not the printer profile, determines the quality mode and the rendering intent to be used in color matching the image to the destination printer. Now that it has a reference to the system profile, the code can obtain the profile's header. It does this by calling the function `CMGetProfileHeader`, specifying the reference it obtained to the system profile.

Using the `kSpeedAndQualityFlagMask` constant it defined earlier, the code clears the quality mode bits of the system profile's `flags` field. Then it sets the quality mode bits to `cmBestMode` to specify best mode quality for color matching. The least significant 2 bits of the `flags` field's high word constitute the quality flag. After setting the quality flag, the code sets the system profile header's `renderingIntent` field to `cmSaturation`.

Now that the code has modified the system profile's header to indicate the user's selections, it calls the `CMSetProfileHeader` function to write the profile header to the profile. Because the driver code intends to use the values the user selected only to color match the image to be printed, it does not permanently preserve the header field changes by calling `CMUpdateProfile` to write the changes to the profile. When the code closes its reference to the system profile after having built the color world, the system profile's header modifications are discarded.

The code calls the `NCWNewColorWorld` function, passing the temporarily modified system profile, to create the color world. It then closes its references to both the system and printer profiles and color matches the image before sending it to the printer. When it no longer needs the color world, the code calls the `CWDisposeColorWorld` function to close the color world and release the memory it uses. Finally, the code tests to ensure that the profile references are closed.

Listing 5-1 Modifying a profile header's quality flag and setting the rendering intent

```
void MySetHeader(void);
CMError MyGetPrinterProfile(CMProfileRef *printerProf);
/* for CM2Header.profileVersion */
#define kMajorVersionMask 0XFF000000
/* two bits used to specify speed & quality */
/* must be shifted left 16 bits in flag's long word */
#define kSpeedAndQualityFlagMask 0X00000003
void MySetHeader(void)
{
    CMError                     cmErr;
    CMProfileRef                sysProf;
    CMAppleProfileHeader        sysHeader;
    CMProfileRef                printerProf;
    CMWorldRef                  cw;

    sysProf = NULL;
    printerProf= NULL;
    cw  = NULL;

    cmErr = CMGetSystemProfile(&sysProf);
    if (cmErr == noErr)
    {
        cmErr = MyGetPrinterProfile(&printerProf);
    }

    if (cmErr == noErr)
    {
        cmErr = CMGetProfileHeader(sysProf, &sysHeader);
    }

    if (cmErr == noErr)
    {
        /* clear the current quality and then set it to best */
        sysHeader.cm2.flags &= ~(kSpeedAndQualityFlagMask << 16);
        sysHeader.cm2.flags |= (cmBestMode << 16);

        /* set rendering intent to saturation */
        sysHeader.cm2.renderingIntent = cmSaturation;
        cmErr = CMSetProfileHeader(sysProf, &sysHeader);
    }

    if (cmErr == noErr)
    {
        cmErr = NCWNewColorWorld(&cw, sysProf, printerProf);
    }

    /* close any open profiles */
    if (sysProf != NULL)
    {
        (void) CMCloseProfile(sysProf);
    }

    if (printerProf != NULL)
    {
        (void) CMCloseProfile(printerProf);
    }
                    .
                    .
                    .
        /* device-driver functions that use the color world to color match
            the image and send it to the printer belong here */
                    .
                    .
                    .
    if (cw != NULL)
    {
        CWDisposeColorWorld(cw);
    }
}
```

Color Matching an Image to Be Printed

The ColorSync Manager provides QuickDraw-specific and general purpose color-matching functions, as described in When Color Matching Occurs (page 38). Printer device drivers usually perform color matching using the general purpose ColorSync Manager functions to match all QuickDraw operations as they pass through the bottleneck routines of the printing grafport.

Note: The general-purpose functions can perform all the operations performed by the QuickDraw-specific functions, but the reverse is not true.

When the stream of QuickDraw data sent to your printer device driver contains a profile embedded using picture comments, your driver should extract the embedded profile using the ColorSync Manager's `CMUnflattenProfile` function. After you extract the profile and open a reference to it, you should create a new color world based on the extracted profile and a profile for your printer. For information on how to extract an embedded profile, see Extracting Profiles Embedded in Pictures (page 73). Creating a Color World to Use With the General Purpose Functions (page 64) describes how to create a color world.

If the QuickDraw data stream does not contain embedded profiles, your driver should use the system profile as the source profile in creating the color world.

You should then match subsequent QuickDraw operations using the color world before sending them to your printer. See Setting Default Profiles (page 33) for information on how the user and how your code can set default profiles.

Developing Color Management Modules

This section gives a brief overview of color management modules (CMMs) and the role a CMM plays in the ColorSync color management system. You should read this section if you are a third-party developer who creates CMMs that ColorSync (versions 2.0 and greater) can use instead of, or in conjunction with, the default CMM.

Before reading this section, you should read Overview of ColorSync (page 27) for a more complete conceptual explanation of how a CMM fits within the ColorSync system. If you are unfamiliar with terms and concepts such as profile, color space, CMM, and color management, or would like to review these topics, you should also read Overview of Color and Color Management Systems (page 13).

At a minimum, a ColorSync-compatible CMM must be able to match colors across color spaces belonging to different base families and check colors expressed in the color gamut of one device against the color gamut of another device.

In addition to the minimum set of requests a CMM must service, a CMM can also implement support for other requests a ColorSync-supportive application or device driver might make. Among the optional services a CMM might provide are verifying if a particular profile contains the base set of required elements for a profile of its type and directing the process of converting profile data embedded in a graphics file to data in an external profile file accessed through a profile reference and vice versa. A CMM can also provide services for PostScript printers by obtaining or deriving from a profile specific data required by PostScript printers for color-matching processes and returning the data in a format that can be sent to the PostScript printer.

This section provides a high-level discussion of the required and optional ColorSync Manager request codes your CMM might be called to handle, and also describes the Component Manager required request codes to which every component must respond.

For complete details on components and their structure, see the chapter Component Manager in Inside Macintosh: More Macintosh Toolbox.

About Color Management Modules

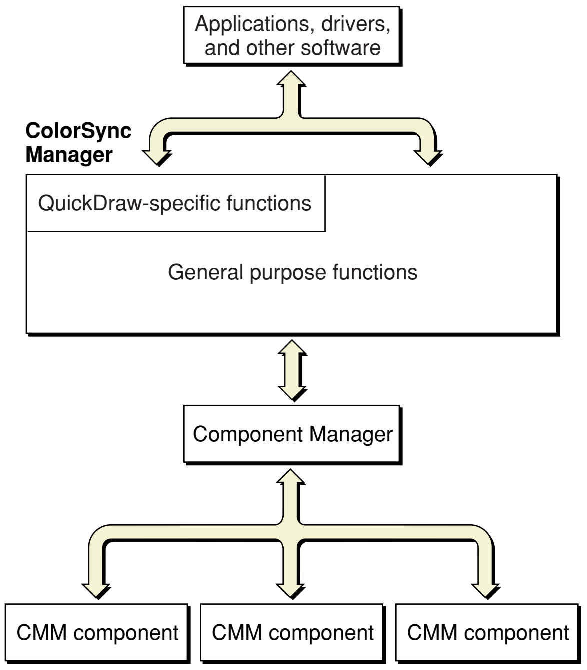
A color management module (CMM) is a component that implements color matching, color gamut checking, and other services and performs these services in response to requests from ColorSync-supportive applications or device drivers.

A CMM component interacts directly with the Component Manager, which calls the CMM on behalf of the ColorSync Manager and the requesting application or driver. When they call ColorSync Manager functions to request color-matching and color gamut-checking services, ColorSync-supportive applications and device drivers specify the profiles to use. These profiles characterize the devices involved; they include information giving the color spaces and the color gamuts of the devices and the preferred CMM to carry out the work. A CMM uses the information contained in these profiles to perform the processing required to service requests. Figure 6-1 (page 122) shows the relationship between a ColorSync-supportive application or driver, the ColorSync Manager, the Component Manager, and one or more available CMM components.

A CMM should support all seven classes of profiles defined by the ICC. For information on the seven classes of profiles, see Profile Class or the International Color Consortium Profile Format Specification, version 2.x or higher. To obtain a copy of the specification, or to get other information about the ICC, visit the ICC Web site at <http://www.color.org/>.

In some cases, a CMM will not be able to convert and match colors directly from the color space of one profile to that of another. Instead, it will need to convert colors to the device-independent color space specified by the profile. A CMM uses device-independent color spaces, or interchange color spaces, to interchange color data from the native color space of one device to the native color space of another device.The profile connection space field of a profile header specifies the interchange color space for that profile. Version 2.x of the ColorSync Manager supports two interchange color spaces: XYZ and Lab.

Figure 6-1 The ColorSync Manager and the Component Manager


Applications, drivers, and other software
ColorSync Manager
QuickDraw-specific functions
General purpose functions
Component Manager
CMM component
CMM component
CMM component

When interchange color spaces are involved, the ColorSync Manager handles the process, which is largely transparent to the CMM. The ColorSync Manager passes to the CMM the correct profiles for color matching. For example, in a case in which both the source and destination profile's CMMs are required to complete the color matching using color space profiles, the ColorSync Manager calls the source profile's CMM with the source profile and an interchange color space profile. Then it calls the destination profile's CMM with an interchange color space profile and the destination profile. The ColorSync Manager assesses the requirements and breaks the process down so that the correct CMM is called with the correct set of profiles. This process is described from the perspective of an application or device driver in How the ColorSync Manager Selects a CMM (page 51).

A CMM uses lookup tables and algorithms for color matching, using one device to preview the color reproduction capabilities of another device, and checking for colors that cannot be reproduced.

Creating a Color Management Module

This section describes how to create a CMM component, including how to respond to required Component Manager and ColorSync Manager requests and optional ColorSync Manager requests. For more detailed information on working with components, see the chapter Component Manager in Inside Macintosh: More Macintosh Toolbox.

Creating a Component Resource for a CMM

A CMM is stored as a component resource. It contains a number of resources, including the standard component resource (a resource of type `'thng'`) required of any Component Manager component. In addition, a CMM must contain code to handle required request codes passed to it by the Component Manager. This includes support for Component Manager required request codes as well as ColorSync Manager required request codes. For an example of the resources your CMM should include, refer to the DemoCMM project available with the ColorSync SDK.

To allow the ColorSync Manager to use your CMM when a profile specifies it as its preferred CMM, your CMM should be located in the Extensions folder, where it will automatically be registered at startup. The file type for component files must be set to `'thng'`.

The Component Resource

The component resource contains all the information needed to register a code resource as a component. Information in the component resource tells the Component Manager where to find the code for the component. As part of the component resource, you must provide a component description record that specifies the component type, subtype, manufacturer, and flags. Here is the data structure for the component description:

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

The following are the key fields of the component description data structure for creating a CMM component:

- The `componentType` field contains a unique 4-byte code specifying the resource type and resource ID of the component's executable code. For your CMM, set this field to `'cmm '`.
- The `componentSubType` field indicates the type of services your CMM provides. You should set this field to your CMM name. This value must match exactly the value specified in the profile header's `CMMType` field. You must register this value with the International Color Consortium (ICC). To obtain information about the ICC, visit the ICC Web site at <http://www.color.org/>.
- The `componentManufacturer` field indicates the creator of the CMM. You may set this field to any value you wish.

- The `componentFlags` field is a 32-bit field that provides additional information about your CMM component. The high-order 8 bits are reserved for definition by the Component Manager. The low-order 24 bits are specific to each component type. You can use these flags to indicate any special capabilities or features of your component.

Note: Values you specify for all fields except the `componentType` field must include at least one uppercase character. Apple Computer reserves values containing all lowercase characters for its own use.

The Extended Component Resource

Since it was first defined, the component resource has been extended to include additional information. That additional information includes the following field for specifying the version of your component:

```
long    componentVersion;   /* version of Component */
```

- The `componentVersion` field indicates the version of the CMM. For related information on specifying the CMM version, see Required Component Manager Request Codes (page 125).

For more information on component data types, see the following files from the Universal Interfaces distributed with development systems for the Mac OS:

- Components.h
- Components.r

How Your CMM Is Called by the Component Manager

Because a CMM is a direct client of the Component Manager, it must conform to the Component Manager's interface requirements, including supporting and responding to required Component Manager calls.

The code for your CMM should be contained in a resource. The Component Manager expects the entry point to this resource to be a function having this format:

```
pascal ComponentResult main(ComponentParameters *params, Handle storage);
```

Whenever the Component Manager receives a request for your CMM, it calls your component's entry point and passes any parameters, along with information about the current connection, in a data structure of type `ComponentParameters`. This entry point must be the first function in your CMM's code segment. The Component Manager also passes a handle to the private storage (if any) associated with the current instance of your component. Here is the component parameters data structure, which is described in detail in Inside Macintosh: More Macintosh Toolbox.

```
struct ComponentParameters {
        unsigned char       flags;
        unsigned char       paramSize;
        short               what;
        long                params[1];
};
```

The first field of the `ComponentParameters` data structure is reserved. The following three fields carry information your CMM needs to perform its processing. The `what` field contains a value that identifies the type of request. The `paramSize` field specifies the size in bytes of the parameters passed from the ColorSync-supportive calling application to your CMM. The parameters themselves are passed in the `params` field.

Required Component Manager Request Codes

At a minimum, your CMM must handle the required Component Manager and required ColorSync Manager request codes. The required Component Manager request codes are defined by these constants:

- `kComponentOpenSelect` (`-1`) Requests that you open an instance of the component. For more information, see Establishing the Environment for a New Component Instance (page 128).
- `kComponentCloseSelect` (`-2`) Requests that you close the component instance. For more information, see Releasing Private Storage and Closing the Component Instance (page 128).
- `kComponentCanDoSelect` (`-3`) Requests that you tell whether your CMM handles a specific request. For more information, see Determining Whether Your CMM Supports a Request (page 128).
- `kComponentVersionSelect` (`-4`) Requests that you return your CMM's version number. For more information, see Providing Your CMM Version Number (page 128). Note that if you provide your version number in an extended component resource, the Component Manager can obtain the version number without having to call your code that handles this request code.

Required ColorSync Manager Request Codes

Your CMM must also be able to handle the required ColorSync Manager request codes defined by these constants:

- `kCMMMatchColors` (1) Requests that you color match the specified colors from one color space to another. For more information, see Matching a List of Colors to the Destination Profile's Color Space (page 130).
- `kCMMCheckColors` (2) Requests that you check the specified colors against the gamut of the destination device whose profile is specified. For more information, see Checking a List of Colors (page 130).
- `kNCMMInit` (6) Requests that you initialize the current component instance of your CMM for a ColorSync Manager 2.x session. For more information, see Initializing the Current Component Instance for a Two-Profile Session (page 129).

Optional ColorSync Manager Request Codes

The Component Manager may also call your CMM with the following ColorSync Manager request codes that are considered optional. A CMM may support these requests, although you are not required to do so.

- `kCMMInit` (0) Requests that you initialize the current component instance of your CMM for a ColorSync 1.0 session. This is a required request code only if your CMM supports ColorSync 1.0 profiles.

- `kCMMMatchPixMap` (3) Requests that you match the colors of a pixel map image to the color gamut of a destination profile, replacing the original pixel colors with their corresponding colors. For more information, see Matching the Colors of a Pixel Map Image (page 134).
- `kCMMCheckPixMap` (4) Requests that you check the colors of a pixel map image against the gamut of a destination device for inclusion and report the results. For more information, see Checking the Colors of a Pixel Map Image (page 134).
- `kCMMConcatenateProfiles` (5) This request code is for backward compatibility with ColorSync 1.0.
- `kCMMConcatInit` (7) Requests that you 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. For more information, see Initializing the Component Instance for a Session Using Concatenated Profiles (page 135).
- `kCMMValidateProfile` (8) Requests that you test a specific profile to determine if the profile contains the minimum set of elements required for a profile of its type. For more information, see Validating That a Profile Meets the Base Content Requirements (page 131).
- `kCMMMatchBitmap` (9) Requests that you match the colors of a source image bitmap to the color gamut of a destination profile. For more information, see Matching the Colors of a Bitmap (page 132).
- `kCMMCheckBitmap` (10) Requests that you check the colors of a source image bitmap against the color gamut of a destination profile. For more information, see Checking the Colors of a Bitmap (page 133).
- `kCMMGetPS2ColorSpace` (11) Requests that you obtain or derive the color space data from a source profile and pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see Obtaining PostScript-Related Data From a Profile (page 136).
- `kCMMGetPS2ColorRenderingIntent` (12) Requests that you obtain the color-rendering intent from the header of a source 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 Obtaining PostScript-Related Data From a Profile (page 136).
- `kCMMGetPS2ColorRendering` (13) Requests that you obtain the rendering intent from the source profile's header, 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 Obtaining PostScript-Related Data From a Profile (page 136).
- `kCMMGetPS2ColorRenderingVMSize` (17) Requests that you obtain or assess the maximum virtual memory (VM) size of the color rendering dictionary (CRD) specified by a destination profile. For more information, see Obtaining the Size of the Color Rendering Dictionary for PostScript Printers (page 137).
- `kCMMFlattenProfile` (14) Requests that you extract profile data from the profile to be flattened and pass the profile data to a function supplied by the calling program. For more information, see Flattening a Profile for Embedding in a Graphics File (page 138).

  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.
- `kCMMUnflattenProfile` (15) Requests that you create a file in the temporary items folder in which to store profile data you receive from a function. The calling program supplies the function. You call this function to obtain the profile data. For more information, see Unflattening a Profile (page 138).

  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.

- `kCMMNewLinkProfile` (16) Requests that you create a single device link profile that includes the profiles passed to you in an array. For more information, see Creating a Device Link Profile and Opening a Reference to It (page 135).
- `kCMMGetNamedColorInfo (70)` Requests that you extract and return named color data from the passed profile reference.
- `kCMMGetNamedColorValue (71)` Requests that you extract and return device and profile connection space (PCS) color values for the specified color name from the passed profile reference.
- `kCMMGetIndNamedColorValue (72)` Requests that you extract and return device and PCS color values for the specified named color index from the passed profile reference.
- `kCMMGetNamedColorIndex (73)` Requests that you extract and return a named color index for the specified color name from the passed profile reference.
- `kCMMGetNamedColorName (74)` Requests that you extract and return a named color name for the specified named color index from the passed profile reference.

Handling Request Codes

When your component receives a request, it should examine the `what` field of the `ComponentParameters` data structure to 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.

Your entry point routine can call a separate subroutine to handle each type of request. *ColorSync Manager Reference* describes the prototypes for functions your CMM must supply to handle the corresponding ColorSync Manager request codes. The entry routine itself can unpack the parameters from the params parameter to pass to its subroutines, or it can call the Component Manager's `CallComponentFunctionWithStorage` routine or `CallComponentFunction` routine to perform these services.

The `CallComponentFunctionWithStorage` function is useful if your CMM uses private storage. When you call this function, you pass it a handle to the storage for this component instance, the `ComponentParameters` data structure, and the address of your subroutine handler. Each time it calls your entry point function, the Component Manager passes to your function the storage handle along with the `ComponentParameters` data structure. For a description of how you associate private storage with a component instance, see Establishing the Environment for a New Component Instance (page 128). The Component Manager's `CallComponentFunctionWithStorage` function extracts the calling application's parameters from the `ComponentParameters` data structure and invokes your function, passing to it the extracted parameters and the private storage handle.

For sample code that illustrates how to respond to the required Component Manager and ColorSync Manager requests, see the DemoCMM project available with the ColorSync SDK. You may also wish to refer to the Apple technical note QT05, Component Manager Version 3.0. This technical note shows how to create a fat component, which is a single component usable for both 68K-based and PowerPC-based systems.

For more information describing how your CMM component should respond to request code calls from the Component Manager, see Creating Components in Inside Macintosh: More Macintosh Toolbox.

Responding to Required Component Manager Request Codes

This section describes some of the processes your CMM can perform in response to the following Component Manager requests that it must handle:

- Establishing the Environment for a New Component Instance (page 128) describes how to handle a `kComponentOpenSelect` request.
- Releasing Private Storage and Closing the Component Instance (page 128) describes how to handle a `kComponentCloseSelect` request.
- Determining Whether Your CMM Supports a Request (page 128) describes how to handle a `kComponentCanDoSelect` request.
- Providing Your CMM Version Number (page 128) describes how to handle a `kComponentVersionSelect` request.

Establishing the Environment for a New Component Instance

When a ColorSync-supportive application or device driver first calls a function that requires the services of your CMM, the Component Manager calls your CMM with a `kComponentOpenSelect` request to open and establish an instance of your component for the calling program. The component instance defines a unique connection between the calling program and your CMM.

In response to this request, you should allocate memory for any private data you require for the connection. You should allocate memory from the current heap zone. It that attempt fails, you should allocate memory from the system heap or the temporary heap. You can use the `SetComponentInstanceStorage` function to associate the allocated memory with the component instance.

For more information on how to respond to this request and open connections to other components, see Creating Components in Inside Macintosh: More Macintosh Toolbox.

Releasing Private Storage and Closing the Component Instance

To call your CMM with a close request, the Component Manager sets the `what` field of the `ComponentParameters` data structure to `kComponentCloseSelect`. In response to this request code, your CMM should dispose of the storage memory associated with the connection.

Determining Whether Your CMM Supports a Request

Before the ColorSync Manager calls your CMM with a request code on behalf of a ColorSync-supportive application or driver that called the corresponding function, the Component Manager calls your CMM with a can do request to determine if your CMM implements support for the request.

To call your CMM with a can do request, the Component Manager sets the `what` field of the `ComponentParameters` data structure to the value `kComponentCanDoSelect`. In response, you should set your CMM entry point function's result to 1 if your CMM supports the request and 0 if it doesn't.

Providing Your CMM Version Number

To call your CMM requesting its version number, the Component Manager sets the `what` field of the `ComponentParameters` data structure to the value `kComponentVersionSelect`. In response, you should set your CMM entry point function's result to the CMM version number. Use the high-order 16 bits to represent the major version and the low-order 16 bits to represent the minor version. The major version should represent the component specification level; the minor version should represent your implementation's version number.

If your CMM supports the ColorSync Manager version 2.x, your CMM should return the constant for the major version defined by the following enumeration when the Component Manager calls your CMM with the `kComponentVersionSelect` request code:

```
enum {
    CMMInterfaceVersion = 1
    };
```

Note that if you provide your version number in an extended component resource, the Component Manager can obtain the version number without having to call your code that handles this request code.

Responding to Required ColorSync Manager Request Codes

This section describes some of the processes your CMM can perform in response to the following ColorSync Manager requests that it must handle:

- Initializing the Current Component Instance for a Two-Profile Session (page 129) describes how to handle the `kNCMMInit` request.
- Matching a List of Colors to the Destination Profile's Color Space (page 130) describes how to handle a `kCMMMatchColors` request.
- Checking a List of Colors (page 130) describes how to handle a `kCMMCheckColors` request.

Initializing the Current Component Instance for a Two-Profile Session

The Component Manager calls your CMM with an initialization request, setting the `what` field of the `ComponentParameters` data structure to `kNCMMInit`. In most cases the Component Manager calls your CMM with an initialization request before it calls your CMM with any other ColorSync Manager requests.

In response to this request, your CMM should call its `NCMInit` initialization subroutine. For a description of the function prototype your initialization subroutine must adhere to, see `NCMInit`.

Using the private storage you allocated in response to the open request, your initialization subroutine should instantiate any private data it needs for the component instance. Before your entry point function returns a function result to the Component Manager, your subroutine should store any profile information it requires. In addition to the standard profile information, you should store the profile header's quality flags setting, the profile size, and the rendering intent. After you return control to the Component Manager, you cannot use the profile references again.

The `kNCMMInit` request gives you the opportunity to examine the profile contents before storing them. If you do not support some aspect of the profile, then you should return an unimplemented error in response to this request. For example, if your CMM does not implement multichannel color support, you should return an unimplemented error at this point.

The Component Manager may call your CMM with the `kNCMMInit` request code multiple times after it calls your CMM with a request to open the CMM. For example, it may call your CMM with an initialization request once with one pair of profiles and then again with another pair of profiles. For each call, you need to reinitialize the storage based on the content of the current profiles.

Your CMM should support all seven classes of profiles defined by the ICC. For the constants used to specify the seven classes of profiles, see *ColorSync Manager Reference*.

Matching a List of Colors to the Destination Profile's Color Space

When a ColorSync-supportive application or device driver calls the `CWMatchColors` function for your CMM to handle, the Component Manager calls your CMM with a color-matching session request, setting the `what` field of the `ComponentParameters` data structure to `kCMMMatchColors` and passing you a list of colors to match. The Component Manager may also call your CMM with this request code to handle other cases, for example, when a ColorSync-supportive program calls the `CWMatchPixMap` function.

Before it calls your CMM with this request, the Component Manager calls your CMM with one of the initialization requests—`kCMMInit`, `kNCMMInit`, or `kCMMConcatInit`—passing to your CMM in the `params` field of the `ComponentParameters` data structure the profiles for the color-matching session.

In response to the `kCMMMatchColors` request, your CMM should call its `CMMatchColors` subroutine by calling the Component Manager's `CallComponentFunctionWithStorage` function and passing it a handle to the storage for this component instance, the `ComponentParameters` data structure, and the address of your `CMMatchColors` subroutine. For a description of the function prototype to which your subroutine must adhere, see `CMMatchColors`.

The parameters passed to your CMM for this request include an array of type `CMColor` containing the list of colors to match and a one-based count of the number of colors in the list.

To handle this request, your CMM must match the source colors in the list to the color gamut of the destination profile, replacing the color value specifications in the `myColors` array with the matched colors specified in the destination profile's data color space. You should use the rendering intent and the quality flag setting of the source profile in matching the colors. For a description of the color list array data structure, see `CMColor`.

Checking a List of Colors

When a ColorSync-supportive application or device driver calls the `CWCheckColors` function for your CMM to handle, the Component Manager calls your CMM with a color gamut-checking session request, setting the `what` field of the `ComponentParameters` data structure to `kCMMCheckColors` and passing you a list of colors to check.

Before the Component Manager calls your CMM with the `kCMMCheckColors` request, it calls your CMM with one of the initialization requests—`kCMMInit`, `kNCMMInit`, or `kCMMConcatInit`—passing to your CMM in the `params` field of the `ComponentParameters` data structure the profiles for the color gamut-checking session.

In response to the `kCMMCheckColors` request, your CMM should call its `CMCheckColors` subroutine. For example, if you use the Component Manager's `CallComponentFunctionWithStorage` function, you pass it a handle to the storage for this component instance, the `ComponentParameters` data structure, and the address of your `CMCheckColors` subroutine. For a description of the function prototype to which your subroutine must adhere, see `CMCheckColors`.

In addition to the handle to the private storage containing the profile data, the `CallComponentFunctionWithStorage` function passes to your `CMCheckColors` subroutine an array of type `CMColor` containing the list of colors to gamut check, a one-based count of the number of colors in the list, and an array of longs.

To handle this request, your CMM should test the given list of colors against the gamut specified by the destination profile to determine whether the colors fall within a destination device's color gamut. For each source color in the list that is out of gamut, you must set the corresponding bit in the result array to 1.

Responding to ColorSync Manager Optional Request Codes

This section describes some of the processes your CMM can perform in response to the optional ColorSync Manager requests if your CMM supports them. Before the Component Manager calls your CMM with any of these requests, it first calls your CMM with a can do request to determine if you support the specific optional request code. This section includes the following:

- Validating That a Profile Meets the Base Content Requirements (page 131) describes how to handle a `kCMMValidateProfile` request.
- Matching the Colors of a Bitmap (page 132) describes how to handle a `kCMMMatchBitmap` request.
- Checking the Colors of a Bitmap (page 133) describes how to handle a `kCMMCheckBitmap` request.
- Matching the Colors of a Pixel Map Image (page 134) describes how to handle the `kCMMMatchPixMap` request.
- Checking the Colors of a Pixel Map Image (page 134) describes how to handle the `kCMMCheckPixMap` request.
- Initializing the Component Instance for a Session Using Concatenated Profiles (page 135) describes how to handle a `kCMMConcatInit` request.
- Creating a Device Link Profile and Opening a Reference to It (page 135) describes how to handle a `kCMMNewLinkProfile` request.
- Obtaining PostScript-Related Data From a Profile (page 136) describes how to handle the `kCMMGetPS2ColorSpace`, `kCMMGetPS2ColorRenderingIntent`, and `kCMMGetPS2ColorRendering` requests.
- Obtaining the Size of the Color Rendering Dictionary for PostScript Printers (page 137) describes how to handle a `kCMMGetPS2ColorRenderingVMSize` request.
- Flattening a Profile for Embedding in a Graphics File (page 138) describes how to handle a `kCMMFlattenProfile` request.
- Unflattening a Profile (page 138) describes how to handle a `kCMMUnflattenProfile` request.
- Supplying Named Color Space Information (page 139) describes how to handle the kCMMGetNamedColorInfo, kCMMGetNamedColorValue, kCMMGetIndNamedColorValue, kCMMGetNamedColorIndex, and kCMMGetNamedColorName requests.

Validating That a Profile Meets the Base Content Requirements

When a ColorSync-supportive application or device-driver calls the `CMValidateProfile` function for your CMM to handle, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to `kCMMValidateProfile` if your CMM supports the request.

In response to this request code, your CMM should call its `CMMValidateProfile` subroutine. One way to do this, for example, is by calling the Component Manager's `CallComponentFunction` function, passing it the `ComponentParameters` data structure and the address of your `CMMValidateProfile` subroutine. To handle this request, you don't need private storage for ColorSync profile information, because the profile reference is passed to your function. However, if your CMM uses private storage for other purposes, you should call the Component Manager's `CallComponentFunctionWithStorage` function. For a description of the function prototype to which your subroutine must adhere, see `CMMValidateProfile`.

The CallComponentFunction function passes to your CMMValidateProfile subroutine a reference to the profile whose contents you must check and a flag whose value you must set to report the results.

To handle this request, your CMM should test the profile contents against the baseline profile elements requirements for a profile of this type as specified by the International Color Consortium. It should determine if the profile contains the minimum set of elements required for its type and set the response flag to true if the profile contains the required elements and false if it doesn't.

To obtain a copy of the International Color Consortium Profile Format Specification, version 2.x, visit the ICC Web site at <http://www.color.org/>.

The ICC also defines optional tags, which may be included in a profile. Your CMM might use these optional elements to optimize or improve its processing. Additionally, a profile might include private tags defined to provide your CMM with processing capability it uses. The profile developer can define these private tags, register the tag signatures with the ICC, and include the tags in a profile.

If your CMM is dependent on optional or private tags, your CMMValidateProfile function should check for the existence of these tags also.

Instead of itself checking the profile for the minimum profile elements requirements for the profile class, your CMMValidateProfile function may use the Component Manager functions to call the default CMM and have it perform the minimum defaults requirements validation.

To call the default CMM when responding to a kCMMValidateProfile request from an application, your CMM can use the standard mechanisms applications use to call a component. For information on these mechanisms, see the chapter Component Manager in Inside Macintosh: More Macintosh Toolbox.

Matching the Colors of a Bitmap

When a ColorSync-supportive application or device driver calls the CWMatchBitMap function for your CMM to handle, the Component Manager calls your CMM with the what field of the ComponentParameters data structure set to kCMMMatchBitmap if your CMM supports the request. If your CMM supports this request code, your CMM should be prepared to receive any of the bitmap types defined by the ColorSync Manager.

In response to this request code, your CMM should call its CMMatchBitmap subroutine. For example, to do this, your CMM may call the Component Manager's CallComponentFunctionWithStorage function, passing it the storage handle for this component instance, the ComponentParameters data structure, and the address of your CMMatchBitmap subroutine. For a description of the function prototype to which your subroutine must adhere, see CMMatchBitmap.

In addition to the storage handle for private storage for this component instance, the CallComponentFunctionWithStorage function passes to your CMMatchBitmap subroutine a pointer to the bitmap containing the source image data whose colors your function must match, a pointer to a callback function supplied by the calling program, a reference constant your subroutine must pass to the callback function when you invoke it, and a pointer to a bitmap in which your function stores the resulting color-matched image.

The callback function supplied by the calling function monitors the color-matching progress as your function matches the bitmap colors. You should call this function at regular intervals. Your CMMatchBitmap function should monitor the progress function for a returned value of true, which indicates that the user interrupted the color-matching process. In this case, you should terminate the color-matching process.

To handle this request, your CMMatchBitmap function must match the colors of the source image bitmap to the color gamut of the destination profile using the profiles specified by a previous kNCMInit, kCMMInit, or kCMMConcatInit request to your CMM for this component instance. You must store the color-matched image in the bitmap result parameter passed to your subroutine. If you are passed a NULL parameter, you must match the bitmap in place.

For a description of the prototype of the callback function supplied by the calling program, see MyCMBitmapCallBackProc.

Checking the Colors of a Bitmap

When a ColorSync-supportive application or device driver calls the CWCheckBitMap function for your CMM to handle, the Component Manager calls your CMM with the what field of the ComponentParameters data structure set to kCMMCheckBitmap if your CMM supports the request. If your CMM supports this request code, your CMM should be prepared to receive any of the bitmap types defined by the ColorSync Manager.

In response to this request code, your CMM should call its CMCheckBitmap subroutine. For example, to do this, your CMM may call the Component Manager's CallComponentFunctionWithStorage function, passing it the storage handle for this component instance, the ComponentParameters data structure, and the address of your CMCheckBitmap subroutine. For a description of the function prototype to which your subroutine must adhere, see MyCMBitmapCallBackProc.

In addition to the storage handle for private storage for this component instance, the CallComponentFunctionWithStorage function passes to your CMCheckBitmap subroutine a pointer to the bitmap containing the source image data whose colors your function must check, a pointer to a callback progress-reporting function supplied by the calling program, a reference constant your subroutine must pass to the callback function when you invoke it, and a pointer to a resulting bitmap whose pixels your subroutine must set to show if the corresponding source color is in or out of gamut. A black pixel (value 1) in the returned bitmap indicates an out-of-gamut color, while a white pixel (value 0) indicates the color is in gamut.

The callback function supplied by the calling function monitors the color gamut-checking progress. You should call this function at regular intervals. Your CMCheckBitmap function should monitor the progress function for a returned value of true, which indicates that the user interrupted the color gamut-checking process. In this case, you should terminate the process.

For a description of the prototype of the callback function supplied by the calling program, see MyCMBitmapCallBackProc.

Using the content of the profiles that you stored at initialization time for this component instance, your CMCheckBitmap subroutine must check the colors of the source image bitmap against the color gamut of the destination profile. If a pixel is out of gamut, your function must set the corresponding pixel in the result image bitmap to 1. The ColorSync Manager returns the resulting bitmap to the calling application or driver to report the outcome of the check.

For complete details on the CMCheckBitmap subroutine parameters and how your CMCheckBitmap subroutine communicates with the callback function, see MyCMBitmapCallBackProc.

Matching the Colors of a Pixel Map Image

When a ColorSync-supportive application or device driver calls the CWMatchPixMap function for your CMM to handle, the Component Manager calls your CMM with the what field of the ComponentParameters data structure set to kCMMMatchPixMap if your CMM supports the request. If your CMM supports this request code, your CMMatchPixMap function should be prepared to receive any of the pixel map types defined by QuickDraw.

In response to this request code, your CMM should call its CMMatchPixMap subroutine. For example, to do this, your CMM may call the Component Manager's CallComponentFunctionWithStorage function, passing it the storage handle for this component instance, the ComponentParameters data structure, and the address of your CMMatchPixMap subroutine. For a description of the function prototype to which your subroutine must adhere, see CMMatchPixMap.

In addition to the storage handle for private storage for this component instance, the CallComponentFunctionWithStorage function passes to your CMMatchPixMap subroutine a pointer to the pixel map containing the source image to match, a pointer to a callback progress-reporting function supplied by the calling program, and a reference constant your subroutine must pass to the callback function when you invoke it.

To handle this request, your CMMatchPixMap subroutine must match the colors of the source pixel map image to the color gamut of the destination profile, replacing the original pixel colors of the source image with their corresponding colors expressed in the data color space of the destination profile. The ColorSync Manager returns the resulting color-matched pixel map to the calling application or driver.

The callback function supplied by the calling function monitors the color-matching progress. You should call this function at regular intervals. Your CMMatchPixMap function should monitor the progress function for a returned value of true, which indicates that the user interrupted the color-matching process. In this case, you should terminate the process.

For a description of the prototype of the callback function supplied by the calling program, see MyCMBitmapCallBackProc.

Checking the Colors of a Pixel Map Image

When a ColorSync-supportive application or device-driver calls the CWCheckPixMap function for your CMM to handle, the Component Manager calls your CMM with the what field of the ComponentParameters data structure set to kCMMCheckPixMap if your CMM supports the request.

In response to this request code, your CMM should call its CMCheckPixMap subroutine. For example, to do this, your CMM may call the Component Manager's CallComponentFunctionWithStorage function, passing it the storage handle for this component instance, the ComponentParameters data structure, and the address of your CMCheckPixMap subroutine. For a description of the function prototype to which your subroutine must adhere, see CMCheckPixMap.

In addition to the storage handle for private storage for this component instance, the CallComponentFunctionWithStorage function passes to your CMCheckPixMap subroutine a pointer to the pixel map containing the source image to check, a QuickDraw bitmap in which to report the color gamut-checking results, a pointer to a callback progress-reporting function supplied by the calling program, and a reference constant your subroutine must pass to the callback function when you invoke it.

Using the content of the profiles passed to you at initialization time, your `CMCheckPixMap` subroutine must check the colors of the source pixel map image against the color gamut of the destination profile to determine if the pixel colors are within the gamut. If a pixel is out of gamut, your subroutine must set to 1 the corresponding pixel of the result bitmap. The ColorSync Manager returns the bitmap showing the color gamut-checking results to the calling application or device driver.

Initializing the Component Instance for a Session Using Concatenated Profiles

When a ColorSync-supportive application or device driver calls the `CWConcatColorWorld` function for your CMM to handle, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to `kCMMConcatInit` if your CMM supports the request.

In response to this request code, your CMM should call its `CMConcatInit` subroutine. For example, to do this, your CMM may call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your `CMConcatInit` subroutine. For a description of the function prototype to which your subroutine must adhere, see `CMConcatInit`.

In addition to the storage handle for private storage for this component instance, the `CallComponentFunctionWithStorage` function passes to your `CMConcatInit` subroutine a pointer to a data structure of type `CMConcatProfileSet` containing an array of profiles to use in a subsequent color-matching or color gamut-checking session. The profiles in the array are in processing order—source through destination. The `profileSet` field of the data structure contains the array. If the profile array contains only one profile, that profile is a device link profile. For a description of the `CMConcatProfileSet` data structure, see `CMConcatProfileSet`.

Using the storage passed to your entry point function in the `CMSession` parameter, your `CMConcatInit` function should initialize any private data your CMM will need for a subsequent color session involving the set of profiles. Before your function returns control to the Component Manager, your subroutine should store any profile information it requires. In addition to the standard profile information, you should store the profile header's quality flags setting, the profile size, and the rendering intent. After you return control to the Component Manager, you cannot use the profile references again.

A color-matching or color gamut-checking session for a set of profiles entails various color transformations among devices in a sequence for which your CMM is responsible. Your CMM may use Component Manager functions to call other CMMs if necessary.

There are special guidelines your CMM must follow in using a set of concatenated profiles for subsequent color-matching or gamut-checking sessions. These guidelines are described in `CMConcatInit`.

Creating a Device Link Profile and Opening a Reference to It

When a ColorSync-supportive application or device driver calls the `CWNewLinkProfile` function for your CMM to handle, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to `kCMMNewLinkProfile` if your CMM supports the request.

In response to this request code, your CMM should call its `CMNewLinkProfile` subroutine. For example, to do this, your CMM may call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your `CMNewLinkProfile` subroutine. For a description of the function prototype to which your subroutine must adhere, see `CMNewLinkProfile`.

In addition to the storage handle for private storage for this component instance, the `CallComponentFunctionWithStorage` function passes to your `CMNewLinkProfile` subroutine a pointer to a data structure of type `CMConcatProfileSet` containing the array of profiles that will make up the device link profile.

To handle this request, your subroutine 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. Your CMM must create a file specification for the device link profile. A device link profile cannot be a temporary profile: that is, you cannot specify a location type of `cmNoProfileBase` for a device link profile. For information on how to specify the file location, see Profile Location Type.

The profiles in the array are in the processing order—source through destination—which you must preserve. After your CMM creates the device link profile, it must open a reference to the profile and return the profile reference along with the location specification.

Obtaining PostScript-Related Data From a Profile

There are three very similar PostScript-related request codes that your CMM may support. Each of these codes requests that your CMM obtain or derive information required by a PostScript printer from the specified profile and pass that information to a function supplied by the calling program.

When a ColorSync-supportive application or device driver calls the high-level function corresponding to the request code and your CMM is specified to handle it, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to the corresponding request code if your CMM supports it. Here are the three high-level functions and their corresponding request codes:

- When the application or device driver calls the `CMGetPS2ColorSpace` function, the Component Manager calls your CMM with a `kCMMGetPS2ColorSpace` request code. To respond to this request, your CMM must obtain the color space data from a source profile and pass the data to a low-level data-transfer function supplied by the calling application or device driver.
- When the application or device driver calls the `CMGetPS2ColorRenderingIntent` function, the Component Manager calls your CMM with a `kCMMGetPS2ColorRenderingIntent`request code. To respond to this request, your CMM must obtain the color rendering intent from the source profile and pass the data to a low-level data-transfer function supplied by the calling application or device driver.
- When the application or device driver calls the `CMGetPS2ColorRendering` function, the Component Manager calls your CMM with a `kCMMGetPS2ColorRendering` request code. To respond to this request, your CMM must obtain the rendering intent from the source profile's header. Then your CMM must obtain or derive the color rendering dictionary for that rendering intent from the destination profile and pass the CRD data to a low-level data-transfer function supplied by the calling application or device driver.

In response to each of these request codes, your CMM should call its subroutine that handles the request. For example, to do this, your CMM may call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your subroutine handler.

For a description of the function prototypes to which your subroutine must adhere for each of these requests, see *ColorSync Manager Reference*.

- For `kCMMGetPS2ColorSpace`, see `CMMGetPS2ColorSpace`
- For `kCMMGetPS2ColorRenderingIntent`, see `CMMGetPS2ColorRenderingIntent`

- For `kCMMGetPS2ColorRendering`, see `CMMGetPS2ColorRendering`.

In addition to the storage handle for private storage for this component instance, the `CallComponentFunctionWithStorage` function passes to your subroutine a reference to the source profile containing the data you must obtain or derive, a pointer to the function supplied by the calling program, and a reference constant that you must pass to the supplied function each time your CMM calls it. For `kCMMGetPS2ColorRendering`, your CMM is also passed a reference to the destination profile.

To handle each of these requests, your subroutine must allocate a `data` buffer in which to pass the particular PostScript-related data to the function supplied by the calling application or driver. Your subroutine must call the supplied function repeatedly until you have passed all the data to it. For a description of the prototype of the application or driver-supplied function, see `MyColorSyncDataTransfer`.

For a description of how each of your subroutines must interact with the calling program's supplied function, see the descriptions of the prototypes for the subroutines in *ColorSync Manager Reference*.

Obtaining the Size of the Color Rendering Dictionary for PostScript Printers

When a ColorSync-supportive application or device driver calls the `CMGetPS2ColorRenderingVMSize` function for your CMM to handle, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to `kCMMGetPS2ColorRenderingVMSize` if your CMM supports the request.

In response to this request code, your CMM should call its `CMMGetPS2ColorRenderingVMSize` subroutine. For example, to do this, your CMM may call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your `CMMGetPS2ColorRenderingVMSize` subroutine. For a description of the function prototype to which your subroutine must adhere, see *ColorSync Manager Reference*.

In addition to the storage handle for global data for this component instance, the `CallComponentFunctionWithStorage` function passes to your `CMMGetPS2ColorRenderingVMSize` subroutine a reference to the source profile identifying the rendering intent and a reference to the destination profile containing the color rendering dictionary (CRD) for the specified rendering intent.

To handle this request, your CMM must obtain or assess and return the maximum VM size for the CRD of the specified rendering intent.

If the destination profile contains the Apple-defined private tag `'psvm'`, described in the next paragraph, then your CMM may read the tag and return the CRD VM size data supplied by this tag for the specified rendering intent. If the destination profile does not contain this tag, then you must assess the VM size of the CRD.

The `CMPS2CRDVMSizeType` data type defines the Apple-defined `'psvm'` optional tag that a printer profile may contain to identify the maximum VM size of a CRD for different rendering intents.

This tag's element data includes an array containing one entry for each rendering intent and its virtual memory size. For a description of the data structures that define the tag's element data, see *ColorSync Manager Reference*.

Flattening a Profile for Embedding in a Graphics File

Flattening refers to transferring a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document. Unflattening refers to transferring from the embedded format to an independent disk file.

Starting With ColorSync 2.5

Starting with ColorSync version 2.5, when a ColorSync-supportive application or device driver calls the `CMFlattenProfile` function, the ColorSync Manager calls the flatten function provided by the calling program or driver directly, without going through the preferred, or any, CMM.

Prior to ColorSync 2.5

Prior to ColorSync version 2.5, when a ColorSync-supportive application or device driver calls the `CMFlattenProfile` function for your CMM to handle, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to `kCMMFlattenProfile` if your CMM supports the request.

In response to this request code, your CMM should call its `CMMFlattenProfile` subroutine. For example, to do this, your CMM may call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your `CMMFlattenProfile` subroutine. For a description of the function prototype to which your subroutine must adhere, see `CMMFlattenProfile`.

In addition to the storage handle for private storage for this component instance, the `CallComponentFunctionWithStorage` function passes to your `CMMFlattenProfile` subroutine a reference to the profile to be flattened, a pointer to a function supplied by the calling program, and a reference constant your subroutine must pass to the calling program's function when you invoke it.

To handle this request, your subroutine must extract the profile data from the profile, allocate a buffer in which to pass the profile data to the supplied function, and pass the profile data to the function, keeping track of the amount of data remaining to pass.

For a description of the prototype of the function supplied by the calling program, see `MyColorSyncDataTransfer`. See also `CMMFlattenProfile` for details on how your `CMMFlattenProfile` subroutine communicates with the function supplied by the calling program.

Unflattening a Profile

Unflattening refers to transferring from the embedded format to an independent disk file. Flattening refers to transferring a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document.

Starting With ColorSync 2.5

Starting with ColorSync version 2.5, when a ColorSync-supportive application or device driver calls the `CMUnflattenProfile` function, the ColorSync Manager calls the unflatten function provided by the calling program or driver directly, without going through the preferred, or any, CMM.

Prior to ColorSync 2.5

Prior to ColorSync version 2.5, when a ColorSync-supportive application or device driver calls the `CMUnflattenProfile` function, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to `kCMMUnflattenProfile`, if your CMM supports that request code.

In response to the `kCMMUnflattenProfile` request code, your CMM should call its `CMMUnflattenProfile` function. To do this, your CMM can call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your `CMMUnflattenProfile` function. For more information, see `CMMUnflattenProfile`.

In addition to the storage handle for private storage for this component instance, the `CallComponentFunctionWithStorage` function passes to your `CMMUnflattenProfile` function a pointer to a function supplied by the calling program and a reference constant. Your function passes the reference constant to the calling program's function when you invoke it. The calling program's function obtains the profile data and returns it to your subroutine. For a more information on the data transfer function, see `MyColorSyncDataTransfer`.

To handle this request, your subroutine must create a file in which to store the profile data. You should create the file in the temporary items folder. Your `CMMUnflattenProfile` subroutine must call the supplied `ColorSyncDataTransfer` function repeatedly to obtain the profile data. Before calling the `ColorSyncDataTransfer` function, your `CMMUnflattenProfile` function must allocate a buffer to hold the returned profile data.

Your `CMMUnflattenProfile` function must identify the profile size and maintain a counter to track the amount of data transferred and the amount of data remaining. This information allows you to determine when to call the `ColorSyncDataTransfer` function for the final time.

Supplying Named Color Space Information

When a ColorSync-supportive application or device driver calls the `CMGetNamedColorInfo` function for your CMM to handle, the Component Manager calls your CMM with the `what` field of the `ComponentParameters` data structure set to kCMMGetNamedColorInfo if your CMM supports the request.

In response to this request code, your CMM should call its CMMGetNamedColorInfo subroutine. To do this, your CMM might call the Component Manager's `CallComponentFunctionWithStorage` function, passing it the storage handle for this component instance, the `ComponentParameters` data structure, and the address of your CMMGetNamedColorInfo subroutine.

The CMMGetNamedColorInfo function returns information about a named color space from its profile reference. For a description of the function prototype to which your subroutine must adhere, see `CMMGetNamedColorInfo`.

A named color profile has a value of `'nmcl'` in the Profile/Device class field of its header. If the source profile passed to your CMMGetNamedColorInfo subroutine is a named color profile, you can extract the necessary information to return in the parameters of the CMMGetNamedColorInfo routine.

Your CMM can obtain named color information as well as profile header information by reading the `namedColor2Tag` tag (signature `'ncl2'`). This tag's element data includes a count of named colors, the number of device channels, and a prefix and suffix for each named color name. The data also includes the named color names themselves, along with profile connection space (PCS) and device color information for each named color. For information on the format of the `namedColor2Tag` tag, see the International Color Consortium Profile Format Specification.

Your CMM responds similarly for other named color requests:

- The `CMGetNamedColorValue` routine generates a kCMMGetNamedColorValue request, which you respond to in your CMMGetNamedColorValue routine. The CMMGetNamedColorValue routine returns device and PCS color values from a named color space profile for a specific color name.
- The `CMGetIndNamedColorValue` routine generates a kCMMGetIndNamedColorValue request, which you respond to in your CMMGetIndNamedColorValue routine. The CMMGetIndNamedColorValue routine returns device and PCS color values from a named color space profile for a specific named color index.
- The CMGetNamedColorIndex routine generates a kCMMGetNamedColorIndex request, which you respond to in your CMMGetNamedColorIndex routine. The CMMGetNamedColorIndex routine returns a named color index from a named color space profile for a specific color name.
- The `CMGetNamedColorName` routine generates a kCMMGetNamedColorName request, which you respond to in your CMMGetNamedColorName routine. The CMMGetNamedColorName routine returns a named color name from a named color space profile for a specific named color index.

Version and Compatibility Information

This section describes the `Gestalt` information, shared library version numbers, CMM version numbers, and ColorSync header files you use with different versions of the ColorSync Manager. It also describes CPU and system requirements.

This section also describes backward compatibility support for ColorSync 1.0 functions, profiles, and CMMs provided by the ColorSync Manager in versions 2.0 and later.

In addition, this section explains how to use the ColorSync Manager for color matching between a ColorSync 1.0 profile and a version 2.x profile.

Version 2.5 of the ColorSync Manager replaces all earlier versions of the product, including ColorSync 2.1, 2.0, and 1.0.

Note: There are no changes to the ColorSync Manager API between version 2.5 and version 2.5.1, so this document is up-to-date for ColorSync 2.5.1.

Although ColorSync 1.0 used a proprietary profile format, the ColorSync Manager provides backward compatibility for applications and device drivers written for ColorSync 1.0. Your application that uses the ColorSync Manager can match, convert, and color check colors using version 2.x profiles or, when necessary, using a combination of version 2.x profiles and ColorSync 1.0 profiles.

Important: Although ColorSync version 2.5 fully supports 1.0 format profiles, this support is not guaranteed to continue in future versions. Apple strongly recommends that developers using the 1.0 format move to the 2.x format.

ColorSync Version Information

This section describes the `Gestalt` information, shared library version numbers, CMM version numbers, CPUs, system versions, and ColorSync header files you use with different versions of the ColorSync Manager. Information is provided in the following sections:

- Gestalt, Shared Library, and CMM Version Information (page 142)
- CPU and System Requirements (page 142)
- ColorSync Header Files (page 143)

For additional version information, see the section ColorSync Versions (page 29) and ColorSync and ICC Profile Format Version Numbers (page 30).

Gestalt, Shared Library, and CMM Version Information

Table 7-1 (page 142) lists the version number for each release of the ColorSync Manager, along with the `Gestalt` version number, shared library version number, and `Gestalt` selector code for that version. Note that only the ColorSync version numbers and `Gestalt` version numbers are unique for each version. For more information on `Gestalt` selectors, see *ColorSync Manager Reference*.

Table 7-1 ColorSync Manager version numbers, with corresponding shared library version numbers and `Gestalt` selectors

ColorSync Version	Gestalt Version	Gestalt Selector	Shared Library Version	Color Management Module (CMM) Version
1.0	$00000100	gestaltColorSync10	$00000000	$00000001
1.0.3	$00000110	gestaltColorSync11	$00000000	$00000001
1.0.4	$00000104	gestaltColorSync104	$00000000	$00000001
1.0.5	$00000105	gestaltColorSync105	$00000000	$00000001
2.0	$00000200	gestaltColorSync20	$02000000	$00010001
2.0.1	$00000200	gestaltColorSync20	$02000000	$00010001
2.1.0	$00000210	gestaltColorSync21	$02100000	$00010001
2.1.1	$00000211	gestaltColorSync21	$02100000	$00010001
2.1.2	$00000212	gestaltColorSync21	$02100000	$00010001
2.5	$00000250	gestaltColorSync25	$02500000	$00010002
2.5.1	$00000251	gestaltColorSync251	$02500000	$00010002

CPU and System Requirements

Table 7-2 lists the CPU and system requirements for each release of the ColorSync Manager.

Table 7-2 ColorSync Manager CPU and system requirements

ColorSync Version	CPU	System Version
1.0, 1.0.3, 1.0.4, 1.0.5	68K or PowerPC	On 68K, requires either System 7.0 or System 6.0.7 with 32-bit QuickDraw, version 1.2.For PowerPC, requires System 7.0.
2.0, 2.0.1, 2.1.0, 2.1.1, 2.1.2	68020 or greater or PowerPC	Requires System 7.0 or greater with Color QuickDraw.
2.5, 2.5.1	68020 or greater or PowerPC	Requires System 7.6.1 or greater.

ColorSync Header Files

Table 7-3 describes the ColorSync Manager header files. Note that some header files are no longer used or are not recommended.

Table 7-3 ColorSync header files

Header File	Description	First Used	Status
CMAcceleration.h	CMM acceleration component interface.	2.0	Not used starting with 2.1.
CMApplication.h	ColorSync Manager functions, constants, and data types for applications, device drivers, and CMMs.	1.0	Supported.
CMCalibrator.h	Interface for developing monitor calibrator plug-ins.	2.5	Supported, but not documented here.
CMComponent.h	Old component interface for CMMs. Replaced by CMMComponent.h.	1.0	Not used starting with 2.0.
CMConversions.h	Component interface for old-style conversion routines.	2.0	Supported, but not recommended starting with 2.1.
CMICCProfile.h	Constants and data types for working with ICC profiles.	1.0	Supported.
CMMComponent.h	Component interface for ColorSync CMMs.	1.0	Supported.
CMPRComponent.h	Component interface for ColorSync 1.0 profile responders.	1.0	Supported, but not recommended starting with 2.0.
CMScriptingPlugin.h	Interface for developing scripting plug-ins.	2.5	Supported, but not documented here.

ColorSync Manager 2.x Backward Compatibility

The following sections describe backward compatibility for ColorSync Manager versions greater than 2.0.

ColorSync 2.1 Support in Version 2.5

Existing code written to use version 2.1 of the ColorSync Manager should continue to work with ColorSync 2.5 without modification. Existing code may operate more efficiently in some cases due to optimizations provided with version 2.5, especially for multiple processors, as described in When ColorSync Uses Multiple Processors (page 45). However, existing code can not take full advantage of some new features; for example, see The Profile Cache and Optimized Searching (page 35).

For a guide to the new features in version 2.5 of the ColorSync Manager, see New Features in ColorSync Manager Version 2.5 (page 151).

ColorSync 2.0 Support in Version 2.1

Existing code written to use version 2.0 of the ColorSync Manager should continue to work with ColorSync 2.1 without modification, although it will not necessarily take advantage of the new features described in New Features in ColorSync Manager Version 2.1 (page 159). For example, code written for ColorSync version 2.0 cannot use the profile identifier, an abbreviated data structure that identifies, and possibly modifies, a profile in memory or on disk. An embedded profile identifier requires much less space than an entire profile.

ColorSync Manager 1.0 Backward Compatibility

The ColorSync Manager continues to fully support the ColorSync 1.0 interface, including the ColorSync 1.0 profile responder. Note, that this support is provided primarily for backward compatibility. If you are writing new code, you should take advantage of the many new features added between version 2.0 and version 2.5. However, existing applications and drivers that use ColorSync 1.0 functions will continue to work properly, as will ColorSync 1.0 profiles, ColorSync 1.0 CMMs, and QuickDraw GX 1.0.

Although newer versions of the ColorSync Manager continue to support use of the profile responder from ColorSync 1.0, this feature is not supported by the ColorSync Manager interface.

ColorSync 1.0 Profile Support

The ColorSync Manager continues to support the use of ColorSync 1.0 profiles. For example, you should always use ColorSync 1.0 functions with ColorSync 1.0 profiles, if possible. For example, always use ColorSync 1.0 functions to match colors between the color gamuts of two devices if both devices have ColorSync 1.0 profiles. The four ColorSync 1.0 functions and their new counterparts are listed in Table 7-4.

However, there are times when you may need to use a ColorSync 1.0 profile with a ColorSync 2.x function. The ColorSync Manager’s backward compatibility allows you to do this. For example, a document containing an image to be color matched may include an embedded ColorSync 1.0 source profile for the image. To match the colors of the source image to a device that has a version 2.x profile, you must use 2.x functions because ColorSync 1.0 functions cannot gain access to a version 2.x profile.

Important: Although ColorSync version 2.5 fully supports 1.0 format profiles, this support is not guaranteed to continue in future versions. Apple strongly recommends that developers using the 1.0 format move to the 2.x format.

One of the main differences between ColorSync 1.0 and 2.x functions is the profile format used. The 2.x functions accommodate ColorSync 1.0 profiles so that you can use those profiles if you must. Before describing how to use a ColorSync 1.0 profile with the 2.x functions, this section explains the differences between the ColorSync 1.0 profile format and the version 2.x profile format defined by the International Color Consortium (ICC) and used by the ColorSync Manager.

ColorSync 1.0 Profiles and Version 2.x Profiles

The ColorSync 1.0 profile format was designed by Apple Computer. This profile is memory resident and follows an internal structure based on tables. Although it is an open format, it is not an industry standard.

The ICC profile format implemented in the ColorSync Manager is significantly different from the profile format implemented for ColorSync 1.0. The version 2.x profile format is specified by the ICC and provides an industry standard that allows for interoperability across platforms and devices. A version 2.x profile created for a particular device can be used on systems running different operating systems.

Because the ColorSync 1.0 and version 2.x profile formats differ, the ColorSync Manager must resolve any compatibility issues involving accessing profiles and color matching between profiles. The next section describes how these profile formats differ.

How ColorSync 1.0 Profiles and Version 2.x Profiles Differ

A ColorSync 1.0 profile is smaller than a version 2.x profile and can therefore reside in memory. It is handle-based. A version 2.x profile as implemented by the ColorSync Manager is commonly file-based, but it can also be memory-based. You use an abstract internal data structure, called a profile reference, to access a version 2.x profile.

A ColorSync 1.0 profile contains a header, a copy of the Apple `CMProfileChromaticities` record, profile response data for the associated device, and a profile name string for use in dialog boxes. Custom profiles may also have additional, private data. ColorSync 1.0 defines the following profile data structure:

```
struct CMProfile {
    CMHeader                    header;
    CMProfileChromaticities     profile;
    CMProfileResponse           response;
    CMIString                   profileName;    /* variable length */
    char                        customData[anyNumber];
                                        /* optional custom CMM data */
};
```

The `response` data fields contain nine tables. The first table is for grayscale values. The next three are red, green, and blue values, followed by three for cyan, magenta, and yellow values. The eighth and ninth tables are for CMYK printers requiring undercolor removal and black generation data.

The ColorSync 1.0 profile header, defined by the data structure CMHeader, and the version 2.x profile header, defined by the structure CM2Header, contain many fields in common. However, some fields in the ColorSync 1.0 profile header reflect its table-based nature, while a version 2.x profile has a tagged-element structure. A version 2.x profile also supports use of lookup table transforms that allow for faster processing.

CMMs and Mixed Profiles

Although version 2.x of the ColorSync Manager supports using a mix of ColorSync 1.0 and version 2.x profiles, the success of a matching session involving a ColorSync 1.0 profile depends on the CMM component performing the process. Third-party CMMs may choose not to support ColorSync 1.0 profiles. The default CMM is able to establish a matching session involving one or more ColorSync 1.0 profiles.

For device link profiles, you must include only version 2.x profiles. You cannot mix ColorSync 1.0 and version 2.x profiles in a device link profile.

Converting a 2.x Profile to the 1.0 Format

The ColorSync Manager provides the CMConvertProfile2to1 function to convert 2.x format profiles to the 1.0 profile format. Because 1.0 and 2.x scanner and monitor profiles generally carry the same required color information, no accuracy is lost in converting from one to the other. With printer profiles, however, some accuracy will be lost by conversion, leading to significantly different results. Because of the possible loss of accuracy in some cases, 2.x to 1.0 profile conversion is not encouraged.

Using Newer Versions of the ColorSync Manager With ColorSync 1.0 Profiles

Despite differences between the version 2.x and ColorSync 1.0 profile formats, you can use most of the ColorSync Manager 2.x functions to gain access to ColorSync 1.0 profiles and their contents and to color match to and from the two disparate profile formats, if necessary. The ColorSync Manager makes this possible.

You can open a reference to a ColorSync 1.0 profile using 2.x functions and special data structures that accommodate both profile styles. You can also match the colors of an image expressed in the color gamut of one device whose characteristics are described by a ColorSync 1.0 profile to the colors within the gamut of another device whose characteristics are described by a version 2.x profile.

Important: If you are color matching between devices that both use ColorSync 1.0 profiles, you should use the ColorSync functions that work with 1.0 profiles for the process.

- which version 2.x functions you cannot use for ColorSync 1.0 profiles
- how you can use the ColorSync Manager with ColorSync 1.0 profiles

ColorSync Manager 2.x Functions Not Supported for ColorSync 1.0 Profiles

You cannot use the ColorSync Manager's `CMUpdateProfile` function to update a ColorSync 1.0 profile. The ColorSync Manager does not provide functions for profile version conversions. This is the domain of profile-building tools and calibration applications.

The ColorSync Manager 2.x versions provide a set of functions to search the ColorSync Profiles folder for specific profiles that meet search criteria. These functions act on version 2.x profiles only. If the ColorSync Profiles folder contains ColorSync 1.0 profiles, these functions do not acknowledge them or return results that include them. The 2.x search functions, which are not supported for ColorSync 1.0 functions, are the CMIterateColorSyncFolder, CMNewProfileSearch, CMUpdateProfileSearch, CMDisposeProfileSearch, CMSearchGetIndProfile, CMSearchGetIndProfileFileSpec, CMProfileIdnetifierFolderSearch, and CMProfileIdentifierListSearch functions.

You cannot use the ColorSync Manager’s `CMUseProfileComment` function to generate automatically the picture comments required to embed a ColorSync 1.0 profile. This function is designed to work with version 2.x profiles only.

Using ColorSync 1.0 Profiles With Newer Versions of the ColorSync Manager

You can use versions 2.0 and higher of the ColorSync Manager to match a document image with an embedded 1.0 source profile to the color gamut of a printer defined by a version 2.x profile. Newer versions of the ColorSync Manager are able to contend with both profile formats.

The sections that follow explain how to obtain a reference to the ColorSync 1.0 profile, get the profile’s header, and get its synthesized tags.

Opening a ColorSync 1.0 Profile

To use a ColorSync 1.0 profile, you must obtain a reference to the profile. Obtaining a reference to the profile is synonymous with opening the profile for your program’s use. If the profile is embedded in a document, you must extract the profile before you can open it.

You can use the `CMOpenProfileFile` function to obtain a reference to a ColorSync 1.0 profile. Other ColorSync Manager functions that you use to gain access to the profile’s contents or perform color matching based on the profile require the profile reference as a parameter.

Obtaining a ColorSync 1.0 Profile Header

After you obtain a reference to a profile, you can gain access to the profile’s contents. To gain access to the contents of any of the fields of a profile header, you must get the entire header. The ColorSync Manager allows you to do this using the `CMGetProfileHeader` function. You pass this function the profile reference and a data structure to hold the returned header. The ColorSync Manager defines the following union of type `CMAppleProfileHeader`, containing variants for ColorSync 1.0 and version 2.x ColorSync profile headers for this purpose:

```
union CMAppleProfileHeader {
    CMHeader        cm1;
    CM2Header       cm2;
};
```

You use the `cm1` variant for a ColorSync 1.0 profile header. You can easily test for the version of a profile header to determine which variant to use because the offset of the header version is at the same place for both ColorSync 1.0 profiles and version 2.x profiles.

Obtaining ColorSync 1.0 Profile Elements

The ColorSync Manager provides four tags to allow you to obtain four ColorSync 1.0 profile elements pointed to from the profile header or contained outside the header. To obtain the profile element, you specify its associated tag signature as a parameter to the `CMGetProfileElement` function along with the profile reference. The ColorSync Manager provides the following enumeration that defines these tags:

```
enum {
        cmCS1ChromTag   = 'chrm',
        cmCS1TRCTag     = 'trc ',
        cmCS1NameTag    = 'name',
        cmCS1CustTag    = 'cust'
};
```

`cmCS1ChromTag`
: Profile chromaticities tag signature. Element data for this tag specifies the XYZ chromaticities for the six primary and secondary colors (red, green, blue, cyan, magenta, and yellow).

`cmCS1TRCTag`
: Profile response data tag signature. Element data for this tag specifies the profile response data for the associated device.

`cmCS1NameTag`
: Profile name string tag signature. Element data for this tag specifies the profile name string. This is an international string consisting of a Macintosh script code followed by a length byte and up to 63 additional bytes composing a text string that identifies the profile.

`cmCS1CustTag`
: Custom tag signature. Element data for this tag specifies the private data for a custom CMM.

Embedding ColorSync 1.0 Profiles

In ColorSync 1.0, picture comment types `cmBeginProfile` and `cmEndProfile` are used to begin and end a picture comment.

The `cmEnableMatching` and `cmDisableMatching` picture comments are used to begin and end color matching in ColorSync 1.0 and in newer versions of the ColorSync Manager.

ColorSync 1.0 Functions With Parallel 2.x Counterparts

Starting with version 2.0, the ColorSync Manager implements new versions of four of the functions supported by ColorSync 1.0. In the new version of these functions, for example, a parameter used to specify a profile takes a profile reference.

It is easy to spot a ColorSync 2.x function that is a new version of a ColorSync 1.0 function, because the function name begins with an uppercase letter N, signifying that it is new. The four ColorSync 1.0 functions and their new counterparts are listed in Table 7-4.

Table 7-4 ColorSync 1.0 functions and their ColorSync Manager counterparts

ColorSync 1.0 function	ColorSync 2.x function
pascal CWNewColorWorld (CMWorldRef *cw, CMProfileHandle src, CMProfileHandle dst);	pascal CMError NCWNewColorWorld (CMWorldRef *cw, CMProfileRef src, CMProfileRef dst);

ColorSync 1.0 function	ColorSync 2.x function
pascal CMError CMBeginMatching (CMProfileHandle src, CMProfileHandle dst, CMMatchRef *myRef);	pascal CMError NCMBeginMatching (CMProfileRef src, CMProfileRef dst, CMMatchRef *myRef);
pascal void CMDrawMatchedPicture (PicHandle myPicture, CMProfileHandle dst, Rect *myRect);	pascal void NCMDrawMatchedPicture (PicHandle myPicture, CMProfileRef dst, Rect *myRect);
pascal CMError CMUseProfileComment (CMProfileHandle profile);	pascal CMError NCMUseProfileComment (CMProfileRef prof, unsigned long flags);

If you are writing a new ColorSync-supportive program, you should always use the new ColorSync Manager functions. The ColorSync 1.0 version of these functions will not be supported indefinitely in new releases of the ColorSync Manager.

What's New

This section lists the new features available with version 2.5 of the ColorSync Manager, provides links to new and revised material in other sections, and summarizes changes to ColorSync functions, data types, and constants. It also contains a brief summary of features that were added for ColorSync 2.1.

Note: There are no changes to the ColorSync Manager API between version 2.5 and version 2.5.1, so this document is up-to-date for ColorSync 2.5.1.

This section includes the following:

- New Features in ColorSync Manager Version 2.5 (page 151) lists the features new to version 2.5 and provides links to new and revised material.
- New and Revised Functions, Data Types, and Constants (page 155) provides tables that include a brief description of all new and changed functions, data types, and constants, as well as links to more detailed descriptions.
- New and Revised Code Listings (page 157) describes new and revised code listings for ColorSync 2.5.
- New Features in ColorSync Manager Version 2.1 (page 159) lists the features new to ColorSync version 2.1.
- Other Color Documentation (page 159) explains where you can find information on earlier versions of ColorSync, and on other color technologies such as the Color Picker Manager.

For related information, see Document Revision History (page 161) and About This Document (page 9).

New Features in ColorSync Manager Version 2.5

Version 2.5 of the ColorSync Manager provides many new or enhanced features. The following sections present a brief overview of these features, with links to detailed information in other sections.

New Profile Folder Location

Earlier versions of ColorSync placed the ColorSync Profiles folder inside the Preferences folder. Version 2.5 places the profiles folder at the first level inside the System folder. For backward compatibility, ColorSync may put an alias to the original folder location inside the new profiles folder.

You can now organize profiles by storing them in one level of subfolders within the profiles folder. You can also store aliases to other profiles and profile folders. Profile searching can find profiles in any of these locations.

For an overview of this and related topics, see:

- Profile Search Locations (page 34)
- Where ColorSync Searches for Profiles (page 34)
- Where ColorSync Does Not Look for Profiles (page 35)
- The function description for `NCMGetProfileLocation`
- Optimized Profile Searching (page 152)

Optimized Profile Searching

ColorSync 2.5 uses a cache file to keep track of currently-installed profiles. A flexible new routine, `CMIterateColorSyncFolder`, takes advantage of the profile cache to perform fast profile searches and provide profile information quickly.

For an overview of this topic, see:

- The Profile Cache and Optimized Searching (page 35)

For related information, including sample code that demonstrates optimized searching, see:

- Performing Optimized Profile Searching (page 81)
- The function description for `CMIterateColorSyncFolder`

Monitor Calibration Framework and Per/Monitor Profiles

ColorSync 2.5 uses the Monitors & Sound control panel to provide a monitor calibration framework and per/monitor profiles. Among the features: you can select a separate profile for each available monitor; you can calibrate monitors and, for each monitor, create one or more color profiles (based on variations in gamma, white point, and so on); Apple provides a default calibration plug-in, but you can create your own calibration plug-in or use third-party versions; you can choose from any available calibrator to create a monitor profile.

For an overview of these features, see:

- Monitor Calibration and Profiles (page 41)

Starting with version 2.5, ColorSync also offers new features for working with displays: you can call ColorSync functions to get or set a monitor profile by AVID; you can use an optional profile tag, which you specify with the `cmVideoCardGammaTag` constant, to provide video card gamma data for a profile—when you call the function `CMSetProfileByAVID`, it retrieves the video card gamma data and sets the video card.

For sample code that uses the function `CMGetProfileByAVID`, see:

- Getting the Profile for the Main Display (page 61)

For an overview of video card gamma, see:

- Video Card Gamma (page 43)

For descriptions of the data types and constants you use with video card data, see *ColorSync Manager Reference*.

Scripting Support

ColorSync 2.5 provides an extensible AppleScript framework that allows users to script many common tasks. Among the features:

- Scriptable operations include setting the system profile, matching an image, and embedding a profile in an image.
- Several sample scripts demonstrate how to automate repetitive tasks.
- The scripting framework uses a plug-in architecture that is fully accessible to third-party scripting plug-ins.

For more information, see:

- Scripting Support (page 43)
    - ❏ Scriptable Properties (page 44)
    - ❏ Scriptable Operations (page 44)
    - ❏ Extending the Scripting Framework (page 44)
    - ❏ Sample Scripts (page 45)

Multiprocessor Support

ColorSync's default Color Matching Module, or CMM, now supports multiple processors for some color matching functions. Multiprocessor support is transparent to your code—it is invoked automatically when the required conditions are met. Matching algorithms take advantage of multiple processors with up to 95% efficiency. As a result, an operation can be performed nearly twice as fast when two processors are available. Performance is scalable.

For more information on this topic, see:

- Multiprocessor Support (page 45)
    - ❏ When ColorSync Uses Multiple Processors (page 45)
    - ❏ Efficiency of ColorSync's Multiprocessor Support (page 45)

Sixteen-bit Channel Support

ColorSync's default Color Matching Module now supports 16-bits-per-channel color spaces. The new formats supported are:

- RBG stored in 48 bits per pixel
- CMYK stored in 64 bits per pixel
- Lab stored in 48 bits per pixel

To make use of these new spaces, you specify one of the following constants in the color space field (`space`) of the `CMBitmap` structure:

```
cmRGB48Space
cmCMYK64Space
cmLAB48Space
```

For more information on these constants, see *ColorSync Manager Reference*.

Flexibility in Choosing CMMs and Default Profiles

The ColorSync control panel, which replaces the ColorSync™ System Profile control panel, now lets you choose a preferred CMM from any CMMs that are present.

Related changes include the following:

- ColorSync previously supported only one default profile—the RGB System profile. Users can now use the ColorSync control panel to set default profiles for RGB and CMYK color spaces as well.
- ColorSync provides functions your code can call to get and set default color space profiles for RGB, CMYK, Lab, and XYZ color spaces.

For more information, see:

- Setting a Preferred CMM (page 37)
- Setting Default Profiles (page 33)
- *ColorSync Manager Reference*

Additional Features

Version 2.5 of the ColorSync Manager ships with the following additional features:

- The Kodak Color Matching Module (available as an install option). Some cross-platform applications use the Kodak Color Management System on the Windows platform. Users working with Macintosh versions of those applications can use the Kodak CMM to ensure consistent output.
- New versions of the ColorSync Photoshop plug-ins that take advantage of ColorSync 2.5. The Filter plug-in is accessible from the Photoshop Filters menu, while the Export and Import filters are accessible from the File menu.
- Commonly-requested profiles, including SWOP (standard web offset press) and sRGB (standardized RGB monitor).
- Support for an optional video card gamma tag in profiles. For more information, see Monitor Calibration Framework and Per/Monitor Profiles (page 152).
- A ColorPicker Manager extension that works with ColorSync 2.x.
- A revised version of the CSDemo application provides sample code that demonstrates how to use many of the new features of ColorSync 2.5.

New and Revised Functions, Data Types, and Constants

The tables in this section provide a brief description of new and changed functions, data types, and constants in ColorSync version 2.5, as well as links to more detailed information.

- Table 8-1 shows new and revised functions.
- Table 8-2 shows new and revised data types.
- Table 8-3 shows new and revised constants.

Table 8-1 New and revised functions in ColorSync 2.5

Function	Version 2.5 Notes
`NCMGetProfileLocation`	New. Obtains either a profile location structure for a specified profile or the size of the location structure for the profile. Has parameter to specify size of location structure.
`CMGetProfileLocation`	Not recommended. Use NCMGetProfileLocation (page 233) instead.
`CMFlattenProfile`	Changed. The ColorSync Manager now calls the transfer function directly, without going through the preferred, or any, CMM.
`CMUnflattenProfile`	Changed. The ColorSync Manager now calls the transfer function directly, without going through the preferred, or any, CMM.
`NCWNewColorWorld`	Changed. Use of the system profile has changed, as described in Setting Default Profiles (page 33). This could affect use of `src` and `dst` parameters.
`CWConcatColorWorld`	Changed. Selection of preferred CMM has changed, as described in Setting a Preferred CMM (page 37) and How the ColorSync Manager Selects a CMM (page 51).
`CWNewLinkProfile`	Changed. Selection of preferred CMM has changed, as described in Setting a Preferred CMM (page 37) and How the ColorSync Manager Selects a CMM (page 51).
`CMGetCWInfo`	Changed. Selection of preferred CMM has changed, as described in Setting a Preferred CMM (page 37) and How the ColorSync Manager Selects a CMM (page 51).
`CWMatchBitmap`	Changed. Now supports additional color space constants: `cmGray16Space`, `cmGrayA32Space`, `cmRGB48Space`, `cmCMYK64Space`, and `cmLAB48Space`.
`NCMBeginMatching`	Changed. Use of the system profile has changed, as described in Setting Default Profiles (page 33). This could affect use of `src` and `dst` parameters.
`NCMDrawMatchedPicture`	Changed. Use of the system profile has changed, as described in Setting Default Profiles (page 33). This could affect use of `dst` parameter.

Function	Version 2.5 Notes
CMGetPreferredCMM	New. Identifies the preferred CMM specified by the ColorSync control panel.
CMGetSystemProfile	Changed. Use of the system profile has changed, as described in Setting Default Profiles (page 33).
CMSetSystemProfile	Changed. Use of the system profile has changed, as described in Setting Default Profiles (page 33).
CMGetDefaultProfileBySpace	New. Gets the default profile for the specified color space.
CMSetDefaultProfileBySpace	New. Sets the default profile for the specified color space.
CMGetProfileByAVID	New. Gets the current profile for a monitor.
CMSetProfileByAVID	New. Sets the current profile for a monitor.
CMGetColorSyncFolderSpec	Changed. The name and location of the profile folder changed, as described in Profile Search Locations (page 34).
CMIterateColorSyncFolder	New. Provides optimized profile searching by iterating over available profiles.
CMNewProfileSearch	Not recommended. Use CMIterateColorSyncFolder instead.
CMUpdateProfileSearch	Not recommended. Use CMIterateColorSyncFolder instead.
CMDisposeProfileSearch	Not recommended. Use CMIterateColorSyncFolder instead.
CMSearchGetIndProfile	Not recommended. Use CMIterateColorSyncFolder instead.
CMSearchGetInd-ProfileFileSpec	Not recommended. Use CMIterateColorSyncFolder instead.
MyProfileIterateProc	New. Application-defined function that the CMIterateColor-SyncFolder function calls once for each found profile file as it iterates over the available profiles.
MyColorSyncDataTransfer	Changed. The ColorSync Manager calls the function directly, without going through the preferred, or any, CMM

Table 8-2 shows new and revised data types.

Table 8-2 New and revised data types in ColorSync 2.5

Data type	Version 2.5 Notes
CMProfileIterateProcPtr	New. Universal procedure pointer definition for application-defined function you pass to the function CMIterateColorSyncFolder.
CMProfileIterateData	New. Provides concise description of key profile data during iteration over available profiles.

Data type	Version 2.5 Notes
`CMSearchRecord`	Not recommended. Use `CMProfileIterateData` instead.
`CMProfileSearchRef`	Not recommended. Use `CMProfileIterateData` instead.
`CMVideoCardGammaType`	New. Optional profile tag for video card gamma.
`CMVideoCardGammaTable`	New. Specifies video card gamma data in table format, based on the specified number of channels, entries per channel, and entry size.
`CMVideoCardGammaFormula`	New. Specifies video card gamma data as a formula, based on specified actual, minimum, and maximum values for red, blue and green gamma.
`CMVideoCardGamma`	New. Specifies video gamma data to store with a video gamma profile tag, in either table or formula format.

Table 8-3 shows new and revised constants.

Table 8-3 New and revised constants in ColorSync 2.5

Constants	Version 2.5 Notes
Color Packing for Color Spaces	Changed. The constants `cm48_16ColorPacking` and `cm64_16ColorPacking` were added.
Abstract Color Space Constants	Changed. The constants `cmRGBASpace` and `cmGrayASpace` were moved from Color Space Constants With Packing Formats.
Color Space Constants With Packing Formats	Changed. The constants `cmRGBASpace` and `cmGrayASpace` were moved to Abstract Color Space Constants.The constants `cmGray16Space`, `cmGrayA32Space`, `cmRGB48Space`, `cmCMYK64Space`, and `cmLAB48Space` were added.
Device Attribute Values for Version 2.x Profiles	Changed. The illustration was revised to show the correct ICC definitions for the `deviceAttributes` field in the `CM2Header` data structure. Unused enums were removed.
Video Card Gamma Tag	New. Specifies the video card gamma tag in a profile.
Video Card Gamma Tag Type	New. Specifies the signature type for a video card gamma profile tag.
Video Card Gamma Storage Type	New. Specifies whether the data in a video card gamma tag is in table or formula format.

New and Revised Code Listings

This section provides a brief description of new and revised code listings.

Table 8-4 New and revised code listings for ColorSync 2.5

Listing	**Version 2.5 Notes**
Listing 4-1 (page 56),Determining if ColorSync 2.5 is available (page 56)	Revised. Checks for version 2.5.
Listing 4-2 (page 59),Opening a reference to a file-based profile (page 59)	Revised. Replaced profLoc.u.file.spec with profLoc.u.fileLoc.spec.
Listing 4-3 (page 60),Poor man's exception handling macro (page 60)	New. Provides the `require` macro for simple error handling.
Listing 4-4 (page 61), Identifying the current system profile (page 61)	Revised. Returns `CMError` instead of `void`. Uses `require` error-handling macro.
Listing 4-5 (page 61), Getting the profile for the main display (page 61)	New. Uses the new `CMGetProfileByAVID` function to get the profile for the main display.
Listing 4-6 (page 63), Matching a picture to a display (page 63)	Revised. Formerly called both `NCMBeginMatching` and `NCMDrawMatchedPicture`. Now calls only the latter. Uses `require` error-handling macro.
Listing 4-6 (page 63), Matching a picture to a display (page 63)	Revised. Formerly called both `CWMatchPixMap` and `CWMatchBitmap`. Now calls only the latter (fixes bug 1669727). Uses `require` error-handling macro.
Listing 4-8 (page 72), Embedding a profile by prepending it before its associated picture (page 72)	Revised. Uses `require` error-handling macro. Disposes of graphics world if necessary on error condition.
Listing 4-9 (page 74), Counting the number of profiles in a picture (page 74)	Revised. Renamed bottleneck procedures for clarity.
Listing 4-10 (page 76), Calling the CMUnflattenProfile function to extract an embedded profile (page 76)	Revised. Uses `require` error-handling macro. Performs cleanup if necessary on error condition.
Listing 4-13 (page 82), An iteration function for profile searching with ColorSync 2.5 (page 82)	New. Provides an iteration function for optimized profile searching with the new `MyProfileIterateProc` function.
Listing 4-14 (page 83), A filter function for profile searching prior to ColorSync 2.5 (page 83)	New. Provides a filter function to perform profile searching with the `CMNewProfileSearch` function that mimics the optimized searching supported by the `MyProfileIterateProc` function.
Listing 4-15 (page 85), Optimized profile searching compatible with previous versions of ColorSync (page 85)	New. Provides sample code that performs an optimized profile search if ColorSync 2.5 is available, but provides a compatible (though not optimized) search if it is not.
Listing 5-1 (page 118), Modifying a profile header's quality flag and setting the rendering intent (page 118)	Revised. Additional comments.

New Features in ColorSync Manager Version 2.1

This section describes new features added to version 2.1 of the ColorSync Manager. These features are documented throughout this document. If you are interested in documentation that covers only version 2.1, see Other Color Documentation (page 159).

- procedure-based profiles: You can specify your own profile-access procedure that ColorSync will call when the profile is created, initialized, opened, read, updated, or closed.
- support for named color spaces: The ColorSync Manager provides data structures and routines for working with named color spaces.
- profile identifiers: The ColorSync Manager defines the profile identifier, an abbreviated data structure that identifies, and possibly modifies, a profile in memory or on disk. An embedded profile identifier requires much less space than an entire profile.
- additional PostScript support: Postscript Level 2 now supports up to four-component color spaces. This allows the creation of device-independent color space definitions that can support calibrated CMYK spaces and provide more flexible support for calibrated scanner and monitor spaces.
- color conversion without components: Color conversion routines are an integral part of the ColorSync Manager and are no longer implemented as a separate component.
- support for new bitmap formats: The ColorSync Manager supports bitmap formats for many additional color spaces, including 24-bit RGB, 32-bit RGB with an alpha last channel, and 24-bit Lab.
- profile reference counts: The ColorSync Manager maintains an internal reference count for each profile reference so that it can efficiently free private memory associated with that profile reference once it is no longer in use.
- profile changed flag: The ColorSync Manager maintains a flag that indicates whether the content of a profile has changed.
- speed and accuracy enhancements: You can use a lookup only flag to skip interpolation and speed up runtime color conversion. You can also disable gamut checking to speed up initialization and reduce profile size.
- revised sample application: A revised version of the CSDemo application provides sample code that demonstrates how to use many of the new features of ColorSync 2.1.

For a guide to the new features in version 2.1 of the ColorSync Manager, see the document What's New in Advanced Color Imaging on the Mac OS, available with the ColorSync 2.1 SDK.

Other Color Documentation

For documentation that covers only features available with ColorSync Manager 2.1 and earlier versions, see Advanced Color Imaging on the Mac OS Revised for ColorSync 2.1 and Advanced Color Imaging Reference Revised for ColorSync 2.1. These documents also describe the Color Picker Manager (Version 2.0), Color Manager, and Palette Manager.

An earlier, paper version of Advanced Color Imaging on the Mac OS, covering ColorSync through version 2.0, was published by Addison-Wesley Publishing Company. It has the catalog number ISBN 0-201-48949-X.

Technote 1100, Color Picker 2.1 describes version 2.1 of the Color Picker Manager. Note that Color Picker Manager version 2.1 works with ColorSync Manager versions 2.0 and greater.

The electronic documents described here are available at <http://developer.apple.com/>.

Document Revision History

This table describes the changes to *Managing Colors With ColorSync in Mac OS 9*.

Date	Notes
2003-02-01	Structured document.
1999-01-01	Last update.

Index