What's New in ColorSync 2.5

Apple Technical Publications
7/15/98

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

Simultaneously published in the United States and Canada.

Contents

Revision History

"What's New" document for ColorSync Manager version 2.5.

Revision	Notes
7/14/98	Added material for video card gamma. Minor elabloration to some other topics.
2/24/98	Added "Working With ColorSync" section to include warnings and minimum requirements. Revised descriptions for `CMIterateColorSyncFolder` function and `CMProfileIterateProcPtr` definition. Minor text changes to "Monitor Calibration and Profiles" section.
1/22/98	Seed release. Provides overview of new features in ColorSync Manager version 2.5, including descriptions of new API.

This document describes new features available with ColorSync Manager Version 2.5.

For information on features available with ColorSync Manager 2.1 and earlier versions, see the electronic book *Advanced Color Imaging on the Mac OS*, which also describes the Color Picker Manager (Version 2.0), Color Manager, and Palette Manager.

Technote 1100, "Color Picker 2.1" describes version 2.1 of the Color Picker Manager.

These documents are available at the Apple Developer World site, which you can reach at <http://developer.apple.com/>.

Working With ColorSync 2.5

This section contains important information you should read before working with ColorSync version 2.5

Minimum Requirements

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.

Warning

To avoid conflict with ColorSync, it is recommended that you remove or turn off any monitor calibration utilities, such as the IMS Twin Turbo control panel, that adjust gamma or modify other calibration values. You should also remove any items in the Startup Items folder than invoke such a utility.

Summary of New Features in ColorSync 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. Later sections describe the new features in greater detail.

New Profile Folder Location

Earlier versions of ColorSync placed the ColorSync Profiles folder inside the Preferences folder. Version 2.5 places that folder at the first level inside the System folder. For backward compatibility, ColorSync may put an alias to the original folder inside the new profiles folder.

For more information, see "Location of the ColorSync Profiles Folder" (page 5).

Subfolders and Aliases

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 includes all profiles in any of these locations.

For more information, see "Profile Search Locations" (page 6).

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 more information, see "Optimized Profile Searching" (page 8).

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 more information, see "Monitor Calibration and Profiles" (page 13).

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 21).

Multiprocessor Support

ColorSync's default Color Matching Module, or CMM, is supplied by Apple Computer and Linotype-Hell. The default 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, see "Multiprocessor Support" (page 23).

Sixteen-bit Channel Support

ColorSync Manager 2.5 and the default CMM supplied by Apple Computer and Linotype-Hell now support 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
```

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.

Other 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 detail, see "Setting Default Color Space Profiles and the Preferred CMM" (page 24).

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 "Video Card Gamma" (page 15).
- A ColorPicker Manager extension that works with ColorSync 2.x.

New Profile Locations

The following sections describe changes in the storage location for profiles in ColorSync 2.5.

Location of the ColorSync Profiles Folder

The ColorSync Profiles folder is now located in the System folder, rather than in the Preferences folder. This protects profiles from deletion should you accidentally or purposefully delete your Preferences folder. More importantly, placement in the System folder will allow the profiles folder to become a "magic" folder, providing the following benefits:

- In future versions of the Mac OS, profiles dragged onto the System folder will automatically be routed to the profiles folder.
- In future versions of the Mac OS, ColorSync will be able to use the Toolbox `FindFolder` routine to find the profiles folder.

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.

Profile Search Locations

With ColorSync 2.5, profile search routines look for profiles in the following locations:

- in the ColorSync Profiles folder in 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 new 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.

Where ColorSync Does Not Look for Profiles

Because profile searching can currently only go two levels deep, the 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).

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.

Obtaining a Profile Location

ColorSync now provides the `NCMGetProfileLocation` function for obtaining a profile location. This function differs from its predecessor, the `CMGetProfileLocation` function, in that you now pass the size of the location

structure to be filled in. You should use the newer version for the following reasons:

- Code using the older version (`CMGetProfileLocation`) may not be as easily ported to other platforms.
- Specifying the profile size with `NCMGetProfileLocation` ensures that the profile location structure can grow, if necessary, in the future.

The `NCMGetProfileLocation` function is defined as follows:

```
pascal CMError NCMGetProfileLocation (
                    CMProfileRef prof,
                    CMProfileLocation * profLoc,
                    unsigned long * locationSize);
```

`prof` A profile reference of type `CMProfileRef`. Before calling `NCMGetProfileLocation`, you set the reference to specify the profile you wish to obtain the location for.

`profLoc` A pointer to a profile location structure, as described in *Advanced Color Imaging on the Mac OS*. If you pass `NULL`, `NCMGetProfileLocation` returns the size of the profile location structure for the profile specified by `prof` in the `locationSize` parameter. If you instead pass a pointer to memory you have allocated for the structure, on return, the structure specifies the location of the profile specified by `prof`.

`locationSize` A pointer to a value of type `long`. If you pass `NULL` for the `profLoc` parameter, on return, `locationSize` contains the size in bytes of the profile location structure for the profile specified by `prof`. If you pass a pointer to a profile location structure in `profLoc`, set `locationSize` to the size of the structure before calling `NCMGetProfileLocation`.

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result code values.

DISCUSSION

The `NCMGetProfileLocation` function is available starting with ColorSync version 2.5. The best way to use `NCMGetProfileLocation` is to call it twice:

1. Pass a reference to the profile to locate in the `prof` parameter and `NULL` for the `profLoc` parameter. `NCMGetProfileLocation` returns the size of the location structure in the `locationSize` parameter.
2. Allocate enough space for a structure of the returned size, then call the function again, passing a pointer in the `profLoc` parameter; on return, the structure specifies the location of the profile.

It is possible to call `NCMGetProfileLocation` just once, using the constant `cmCurrentProfileLocationSize` for the size of the allocated profile location structure and passing the same constant for the `locationSize` parameter. The constant `cmCurrentProfileLocationSize` may change in the future, but will be consistent within the set of headers you build your application with. However, if the size of the `CMProfileLocation` structure changes in a future version of ColorSync and you do not rebuild your application, `NCMGetProfileLocation` may return an error.

Optimized Profile Searching

The following sections describe changes in the way ColorSync 2.5 stores and manages profile information.

The Profile Cache

ColorSync 2.5 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.

Note, however, that calls to the standard ColorSync search routines 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.

The next section describes a new routine added to the ColorSync API to take full advantage of the profile cache.

Optimized Profile Searching With CMIterateColorSyncFolder

A flexible new routine, `CMIterateColorSyncFolder`, takes advantage of the profile cache to provide truly optimized searching and quick access to profile information.

CMIterateColorSyncFolder

The `CMIterateColorSyncFolder` routine iterates over the available profiles. It is defined as follows:

```
pascal CMError CMIterateColorSyncFolder (
                    CMProfileIterateUPP proc,
                    unsigned long * seed,
                    unsigned long * count,
                    void * refCon);
```

`proc`	A universal procedure pointer of type `CMProfileIterateUPP`, which is described in "CMProfileIterateProcPtr" (page 11). If you do not wish to receive callbacks, pass `nil` for this parameter. Otherwise, pass a pointer to your callback routine.
`seed`	A pointer to a value of type `long`. The first time you call `CMIterateColorSyncFolder`, you typically set the value to 0. In subsequent calls, you set the value to the seed value obtained from the previous call. ColorSync uses the value in determining whether to call your callback routine, as described in the discussion for this function. On return, the value is the current seed for the profile cache (unless you pass `nil`, as described in the discussion).
`count`	A pointer to a value of type `long`. On return, the value is the number of available profiles. `CMIterateColorSyncFolder` provides the number of profiles even when no iteration occurs (unless you pass `nil`, as described in the discussion below). To determine the count alone, without iteration, call `CMIterateColorSyncFolder` and pass a value of `nil` for all parameters except `count`.
`refCon`	A pointer to arbitrary data, supplied by you, that ColorSync passes to your callback routine. If you pass `nil` for this parameter, ColorSync passes `nil` to your callback routine.
function result	A result code of type `CMError`. If your callback function returns an error, `CMIterateColorSyncFolder` stops iterating and returns the error value to its caller (presumably your code). See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result code values.

DISCUSSION

When your application needs information about the currently available profiles, it calls the `CMIterateColorSyncFolder` routine, which in turn calls your callback routine once for each profile. Even though there may be many profiles available, ColorSync can use its profile cache to return profile information quickly, and (if the cache is valid) without having to open any profiles. For each profile, ColorSync returns the profile header, script code, name, and location. As a result, your routine may be able to perform its function, such as building a list of profiles to display in a pop-up menu, without further effort (such as opening a file-based profile).

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

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

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

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

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

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

CMProfileIterateProcPtr

The universal procedure pointer callback routine passed as a parameter to `CMIterateColorSyncFolder` is defined as follows:

```
pascal OSErr (*CMProfileIterateProcPtr )
                    (CMProfileIterateData *iterateData,
                     void *refCon);
```

`iterateData` A pointer to a structure of type `CMProfileIterateData` (defined in the next section). On return, the structure contains profile information for the current profile (as the `CMIterateColorSyncFolder` routine iterates over all available profiles).

`refCon` A pointer to arbitrary data you pass to the `CMIterateColorSyncFolder` routine and it, in turn, passes to your callback routine.

callback return value
A result code of type `CMError`. If your callback function returns an error, `CMIterateColorSyncFolder` stops iterating and returns the error value to its caller (presumably your code). See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.

DISCUSSION

When you call `CMIterateColorSyncFolder`, you pass a universal procedure pointer of type `CMProfileIterateProcPtr` that points to a function you provide. Your function definition is based on this definition of `CMProfileIterateProcPtr`.

CMProfileIterateData

The ColorSync Manager defines the `CMProfileIterateData` structure to provide your `CMProfileIterateProcPtr` (page 11) callback routine with a description of a profile during an iteration through the available (or specified) profiles. The structure is defined as follows:

```
struct CMProfileIterateData {
    unsigned long   dataVersion;    /* cmProfileIterateDataVersion1 */
    CM2Header       header;
    ScriptCode      code;
    Str255          name;
    CMProfileLocation location;
};
typedef struct CMProfileIterateData CMProfileIterateData;
```

`dataVersion` A value identifying the version of the structure. Currently set to `cmProfileIterateDataVersion1`.

`header`	A ColorSync version 2.x profile header structure, containing information such as the profile size, type, version, and so on.
`code`	A script code identifying the script system used for the profile description.
`name`	The profile name.
`location`	A structure specifying the profile location. With ColorSync 2.5, the location is always file-based, but that may not be true for future versions. Your code should always verify that the location structure contains a file specification before attempting to use it.

Monitor Calibration and Profiles

Ever 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 a calibration application to generate a new ColorSync profile for the new monitor settings.

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 1 (page 14) 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.

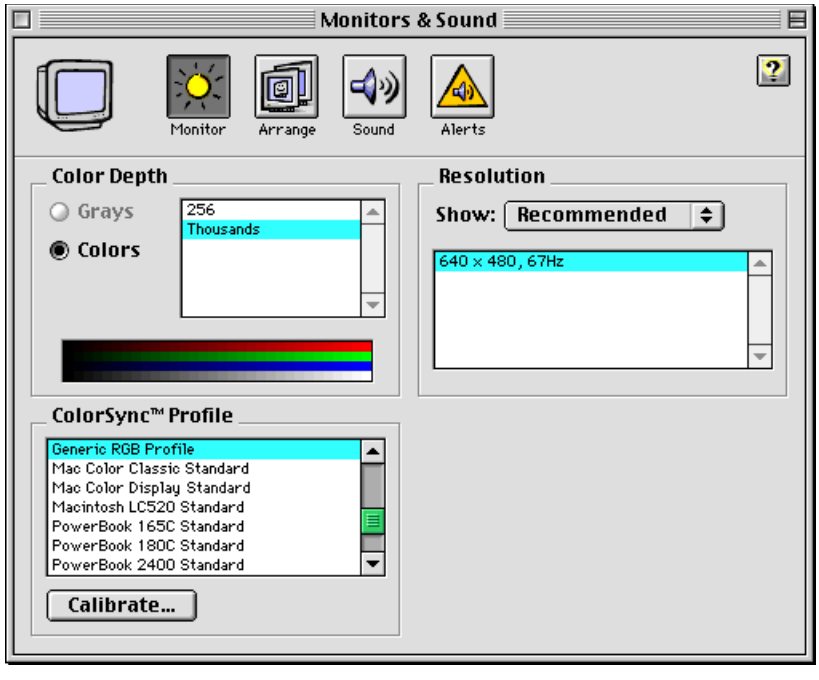
Because Monitors & Sound displays a panel for each available monitor, a user can also select, for each monitor, a separate profile from the list of available profiles. For information on how to set monitor profiles in your code, see "Monitor Profile Functions" (page 29).

The Calibrate button provides the launching point for 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. AppleVision and Apple ColorSync monitors are self-calibrating, so you will not see a Calibrate button for these monitors, unless there is at least one 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.

Figure 1 Monitors & Sound Control Panel for ColorSync 2.5



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 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'`.

If you plan to create a monitor calibration plug-in, you should read the section "Video Card Gamma" (page 15).

Video Card Gamma

ColorSync now supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. When you call the function `CMSetProfileByAVID` (page 29) 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. If you provide monitor calibration software, you should include the video card gamma tag in the profiles you create. See "Monitor Calibration and Profiles" (page 13) for more information on ColorSync's support for monitor calibration.

IMPORTANT

The function `CMSetSystemProfile` (page 274) does not retrieve video card gamma data to set the video card. ▲

Video Card Gamma Constants

The following sections describe the constants you use to work with the video card gamma profile tag.

Video Card Gamma Tag

When you create a tag to store video card gamma data in a profile, you use the `cmVideoCardGammaTag` constant to specify the tag.

```
enum
{
    . . . ,
    cmVideoCardGammaTag = FOUR_CHAR_CODE('vcgt')
};
```

Enumerator descriptions

`cmVideoCardGammaTag`
Constant for profile tag that specifies video card gamma information.

Video Card Gamma Tag Type

You use the `cmSigVideoCardGammaType` constant to specify the signature type for a video card gamma tag. That is, you use this constant to set the `typeDescriptor` field of the `CMVideoCardGammaType` (page 17) structure. There is currently only one type possible for a video card gamma tag.

```
enum
{
    cmSigVideoCardGammaType = FOUR_CHAR_CODE('vcgt')
};
```

Enumerator descriptions

`cmSigVideoCardGammaType`
Constant that specifies video card gamma type signature in a video card gamma profile tag.

Video Card Gamma Storage Type

A video card gamma profile tag can store gamma data either as a formula or as a table of values. You use a storage type constant to specify which data storage type the tag uses.

IMPORTANT

If the video card uses a different format than the format you specify, ColorSync will adapt the data you supply to match the format the card expects. ▲

```
enum {
    cmVideoCardGammaTableType = 0,
    cmVideoCardGammaFormulaType = 1,
};
```

Enumerator descriptions

`cmVideoCardGammaTableType`
The video card gamma data is stored in a table format. See "CMVideoCardGammaTable" (page 18) for a description of the table format.

`cmVideoCardGammaFormulaType`
The video card gamma tag data is stored as a formula. See "CMVideoCardGammaFormula" (page 19) for a description of the formula format.

Video Card Gamma Data Types

The following sections describe data types you use to work with the video card gamma profile tag.

CMVideoCardGammaType

The ColorSync Manager defines the `CMVideoCardGammaType` data structure to specify a video card gamma profile tag.

```
struct CMVideoCardGammaType
{
    OSType              typeDescriptor;
    unsigned long       reserved;
    CMVideoCardGamma    gamma;
};
typedef struct CMVideoCardGammaType CMVideoCardGammaType;
```

Field descriptions

`typeDescriptor` The signature type for a video card gamma tag. There is currently only one type possible, `cmSigVideoCardGammaType`.

`reserved` This field is reserved.

`gamma` A structure that specifies the video card gamma data for the profile tag, as described in "CMVideoCardGamma" (page 20).

CMVideoCardGammaTable

The ColorSync Manager defines the `CMVideoCardGammaTable` data structure to specify video card gamma data in table format. You specify the number of channels, the number of entries per channel, and the size of each entry. The last field in the structure is an array of size one that serves as the start of the table data. The actual size of the array is equal to the number of channels times the number of entries times the size of each entry.

```
struct CMVideoCardGammaTable
{
    unsigned short      channels;
    unsigned short      entryCount;
    unsigned short      entrySize;
    char                data[1];
};
typedef struct CMVideoCardGammaTable CMVideoCardGammaTable;
```

Field descriptions

`channels` Number of gamma channels (1 or 3). If `channels` is set to 1 then the red, green, and blue lookup tables (LUTs) of the video card will be loaded with the same data. If `channels` is

	set to 3, then if the video card supports separate red, green, and blue LUTs, then the video card LUTs will be loaded with the data for the three channels from the `data` array.
`entryCount`	Number of entries per channel (1-based). The number of entries must be greater than or equal to 2.
`entrySize`	Size in bytes of each entry.
`data`	Variable-sized array of data. The size of the data is equal to `channels * entryCount * entrySize`.

CMVideoCardGammaFormula

The ColorSync Manager defines the `CMVideoCardGammaFormula` data structure to specify video card gamma data by providing three values each for red, blue and green gamma. The values represent the actual gamma, the minimum gamma, and the maximum gamma for each color. Specifying video gamma information by formula takes less space than specifying it with a table, but the results may be less precise.

```
struct CMVideoCardGammaFormula {
    Fixed           redGamma;
    Fixed           redMin;
    Fixed           redMax;
    Fixed           greenGamma;
    Fixed           greenMin;
    Fixed           greenMax;
    Fixed           blueGamma;
    Fixed           blueMin;
    Fixed           blueMax;
};
```

Field descriptions

`redGamma`	The gamma value for red. It must be greater than 0.0.
`redMin`	The minimum gamma value for red. It must be greater than 0.0 and less than 1.0.
`redMax`	The maximum gamma value for red. It must be greater than 0.0 and less than 1.0.
`greenGamma`	The gamma value for green. It must be greater than 0.0.

`greenMin`	The minimum gamma value for green. It must be greater than 0.0 and less than 1.0.
`greenMax`	The maximum gamma value for green. It must be greater than 0.0 and less than 1.0.
`blueGamma`	The gamma value for blue. It must be greater than 0.0.
`blueMin`	The minimum gamma value for blue. It must be greater than 0.0 and less than 1.0.
`blueMax`	The maximum gamma value for blue. It must be greater than 0.0 and less than 1.0.

CMVideoCardGamma

The ColorSync Manager defines the `CMVideoCardGamma` data structure to specify the video gamma data to store with a video gamma profile tag. The structure is a union that can store data in either table or formula format.

```
struct CMVideoCardGamma
{
    unsigned long                   tagType;
    union
    {
        CMVideoCardGammaTable       table;
        CMVideoCardGammaFormula     formula;
    }                               u;
};
typedef struct CMVideoCardGamma CMVideoCardGamma;
```

Field descriptions

`tagType`	A "Video Card Gamma Storage Type" (page 17) constant that specifies the format of the data currently stored in the union. To determine the type of structure present in a specific instance of the `CMVideoCardGamma` structure, you test this union tag. When storing video card gamma data, you set `tagType` to a constant value that identifies the structure type.
`table`	A structure of type `CMVideoCardGammaTable`. If the `tagType` field has the value `cmVideoCardGammaTableType`, the

	`CMVideoCardGamma` structure's union field should be treated as a table, as described in "CMVideoCardGammaTable" (page 18).
`formula`	A structure of type `CMVideoCardGammaFormula`. If the `tagType` field has the value `cmVideoCardGammaFormulaType`, the `CMVideoCardGamma` structure's union field represents a formula, as described in "CMVideoCardGammaFormula" (page 19).

Scripting Support

ColorSync 2.5 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.
- Embedding a profile in an image.

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

With ColorSync version 2.5, the default ColorSync CMM supplied by Apple Computer and Linotype-Hell 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 low-level `CWMatchPixmap` and `CWMatchBitmap` calls, or when those calls are invoked as a result of a call to the high-level QuickDraw matching routines (`NCMBeginMatching` and so on).

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 will provide proportionally equal improvement.

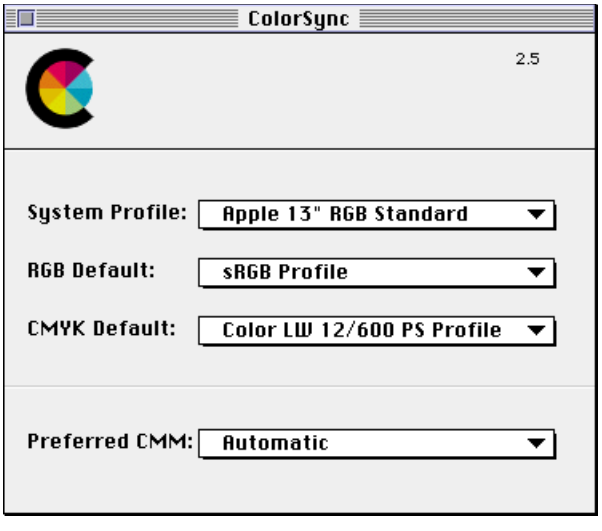
Setting Default Color Space Profiles and the Preferred CMM

ColorSync version 2.5 provides new flexibility for specifying a preferred Color Matching Module (CMM) and default color space profiles.

The ColorSync Control Panel

The ColorSync control panel replaces the ColorSync™ System Profile control panel. The ColorSync panel, with its default settings, is shown in Figure 2.

Figure 2 The ColorSync control panel



Setting Default Color Space Profiles

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. For information on how to get and set the system profile, and how to get and set default profiles for the RGB, CMYK, Lab, and XYZ color spaces, in your code, see "Default Color Space and System Profile Functions" (page 27).

Setting the Preferred CMM

The ColorSync control panel lets you choose a preferred CMM from any CMMs that are present (registered with the Component Manager).

If you choose a preferred CMM with the ColorSync control panel, and if that CMM is available, ColorSync will use that CMM for all color conversion and matching operations. If you specify "Automatic" instead, or if your specified CMM is no longer present or cannot provide the required matching service, ColorSync follows an algorithm described in *Advanced Color Imaging on the Mac OS* to determine which available CMM to use for matching.

Your code can call the `CMGetPreferredCMM` function to get the preferred CMM. The function is defined as follows:

```
pascal CMError CMGetPreferredCMM (
                    OSType *cmmType,
                    Boolean *preferredCMMnotfound)
```

`cmmType` A pointer to an `OSType`. On return, the component subtype for the preferred CMM. For example, the subtype for the default CMM provided by Apple is `'appl'` and the subtype for the Kodak CMM is `'KCMS'`. A return value of nil indicates the preferred CMM in the ColorSync control panel is set to Automatic.

`preferredCMMnotfound`
A pointer to a Boolean flag for whether the preferred CMM was found. On return, has the value `true` if the CMM was found. `false` if it was not.

function result A result code of type `OSErr`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes. Always check the function result before using any information returned in the function parameters.

The `CMGetPreferredCMM` function returns in the `cmmType` parameter a value that identifies the preferred CMM the user last specified in the ColorSync control panel. `CMGetPreferredCMM` returns `true` in the `preferredCMMnotfound` parameter if the preferred CMM is currently available and `false` if it is not. For example, a user may specify a preferred CMM in the ColorSync control panel, then reboot with extensions off. ColorSync does not change the preferred CMM setting when the preferred CMM is not available.

Changes to System Profile and Monitor Profile Operations

With version 2.5, ColorSync has removed certain system profile and monitor profile restrictions. A little history can help explain these restrictions and the new, more flexible approach.

Prior to version 2.5, ColorSync provided the ColorSync™ System Profile control panel. With that panel, a user could only select an RGB profile for the system profile. The system profile was used for the default display profile, as well as for operations on RGB images that resulted in the need for a default profile (such as when the image itself did not specify a profile). So the system profile was used for two dissimilar functions, which led to 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 profile to the screen, no matching occurs because the source and destination profiles are the same (system) profile.

More Robust User Interface

With version 2.5, the ColorSync control panel replaces the ColorSync™ System Profile control panel. The ColorSync panel, with its default settings, is shown in Figure 2 (page 24). The new panel lets the user specify profiles for the system profile (for backward compatibility), as well as for RGB and CMYK color

spaces. As a result, the link between the system profile and the default profile for the RGB color space is broken, and a user can specify separate default profiles for different color spaces. ColorSync also provides functions your code can use to set default profiles, including profiles for Lab and XYZ spaces. These functions are shown in "Default Color Space and System Profile Functions" (page 27).

With version 2.5, a user can now use the Monitors & Sound control panel to specify a profile for each monitor. ColorSync also provides functions your code can use to specify a profile for each monitor. These functions are shown in "Monitor Profile Functions" (page 29).

Default Color Space and System Profile Functions

ColorSync provides the following functions your code can call to get and set default profiles for RGB, CMYK, Lab, and XYZ color spaces, and to get and set the system profile. For example, if your application doesn't want to present a user interface for setting these profiles, you can use these calls.

CMSetSystemProfile

Sets the system profile.

```
pascal CMError CMSetSystemProfile (
                    const FSSpec * profileFileSpec);
```

`FSSpec` A pointer to a file specification structure. Before calling `CMSetSystemProfile`, set the structure to specify the desired system profile.

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.

CMGetSystemProfile

Gets the system profile.

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

`prof` A pointer to a structure of type `CMProfileRef`. On return, a reference to the current system profile.

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.

CMSetDefaultProfileBySpace

Sets the default profile for a color space.

```
pascal CMError CMSetDefaultProfileBySpace (
                    OSType dataColorSpace,
                    CMProfileRef prof);
```

`dataColorSpace`
Specifies the color space for which to set the default profile. You specify one of the currently-supported values, `cmRGBData`, `cmCMYKData`, `cmLabData`, or `cmXYZData`.

`prof` A profile reference. Before calling `CMSetDefaultProfileBySpace`, set the reference to specify the default profile for the color space.

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.

CMGetDefaultProfileBySpace

Gets the default profile for a color space.

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

`dataColorSpace`
Specifies the color space for which to get the default profile. You specify one of the currently-supported values, `cmRGBData`, `cmCMYKData`, `cmLabData`, or `cmXYZData`.

`prof` A pointer to a profile reference. On return, the reference specifies the current profile for the color space specified by `dataColorSpace`.

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.

Monitor Profile Functions

ColorSync provides the following functions your code can call to get and set the profile for each monitor. These routines work with the `AVIDType` data type, which is defined by the Display Manager and used to specify a device such as a monitor.

CMSetProfileByAVID

Sets the profile for a monitor. If the profile contains an optional profile tag for video card gamma, ColorSync will extract the tag from the profile and set the video card based on the tag, as described in "Video Card Gamma" (page 15).

```
pascal CMError CMSetProfileByAVID (
                    AVIDType theAVID,
                    CMProfileRef prof);
```

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

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

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.

CMGetProfileByAVID

Gets the current profile for a monitor.

```
pascal CMError CMGetProfileByAVID (
                    AVIDType theAVID,
                    CMProfileRef *prof);
```

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

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

function result A result code of type `CMError`. See *Advanced Color Imaging on the Mac OS* for a list of ColorSync-specific result codes.