Technical Note TN2035
ColorSync on Mac OS X

CONTENTS

Overview

Colorsync and Quartz

Profiles on Mac OS X

CMMs on Mac OS X

ColorSync Preferences and Supporting APIs

ColorSync Device Support

ColorSync Changes for Mac OS 10.2

References

Downloadables

ColorSync has a new role in Mac OS X. ColorSync on Mac OS X is fundamentally integrated into the operating system, whereas on previous versions of the operating system it was an optional install. Color management is very important these days to the customer experience. The Mac OS X graphics environment Quartz takes advantage of ColorSync in great detail, plus ColorSync is used throughout the print-path to provide color matching services for printer drivers.

[Feb 10 2003]

Overview

There are a wide variety of imaging devices in the market today, such as digital cameras, color scanners, color printers, etc., and each device represents colors in very different ways. They each have different color spaces and different gamuts (ranges of colors). This, of course, makes it very difficult for a document acquired on one device to be rendered correctly on another device. The solution to this problem is ColorSync, a complete color management system, designed to provide consistent color across devices.

The two key foundations upon which ColorSync is built are International Color Consortium (ICC) profiles and Color Management Modules (CMMs).

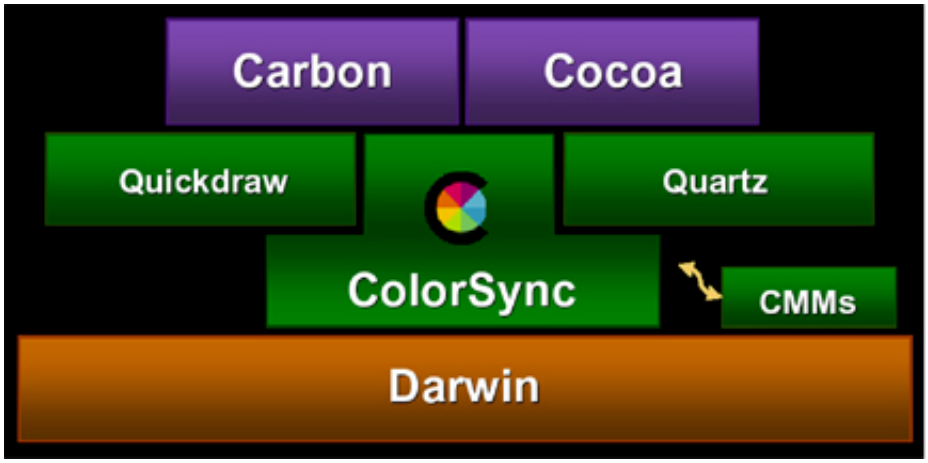
ICC Profiles

ICC profiles are a cross-platform file format defined by the ICC. They are documents containing data that describe how to transform colors from device color space to an intermediate color space. This file format allows for the description of a wide variety of devices. The data format has been designed to be quite flexible, and can be extended by developers via optional tags. Furthermore, the format is an evolving format, with ongoing improvements.

CMM

If you think of profiles as the data used for color management, CMMs are the code. CMMs provide the mathematical engine to perform the profile-to-profile transformations. Apple ships a default CMM, the Apple CMM, as part of ColorSync. This is the same default CMM Apple ships on Mac OS 9. However, we've designed the system to be open and expandable, so third parties can developer their own CMMs if they want to provide their own alternate methods for color transformations.

Here's a diagram showing how ColorSync fits into the Mac OS X architecture:



The ColorSync 2.0 and greater APIs are fully supported on Mac OS X. This means it should be very easy for developers to port their applications to Mac OS X. There are also some new APIs in ColorSync on Mac OS X which provide some enhancements you may want to take advantage of. Also, Quartz and the printing model in Mac OS X take full advantage of ColorSync.

Back to top

ColorSync and Quartz

On Mac OS X, ColorSync has been assigned a new role encompassing more than just the standard application services. ColorSync is now used to provide color management to other system components. One of these components is Quartz, Apple's new graphics system based on the PDF imaging model. Quartz has created a new paradigm for color management, which offers alternative access to ColorSync functionality.

ColorSync And Quartz Color Management

Our goals here were to integrate graphics and color management services, satisfying some basic requirements. First, Quartz needed the ability to composite different color spaces and opacity. To meet this requirement, ColorSync is used to convert data from many different color spaces into the one space selected by Quartz as a working space for a compositer.

A second requirement focused on the needs of the developer, and is based on the fact all applications using Quartz will have to work with color management. This also needed to be a scalable solution, which would serve the needs of a wide range of applications. On the one hand, the involvement of an application in color management can be very minimal, limited to the use of some predefined default settings provided by Quartz. On the other hand, an application can have the same full control over color management as it does when accessing ColorSync directly. Other requirements include color accuracy, performance and compatablilty with PDF.

In simple terms, Quartz color management can be described as being built around ColorSync, which is used as an engine to process PDF color data produced by Quartz.

PDF Color Model

To better understand its design, here's some of the basic concepts pertaining to color in PDF and ColorSync:

PDF Color Definitiion

In PDF, color is defined by one of the known color spaces - Device, Calibrated or ICC based:

- /DeviceGray, /DeviceRGB, /DeviceCMYK
- /CalGray, /CalRGB, /CalLab

- /ICCBased

This list essentially reflects the history of color management. Initially, only device color spaces were supported. Based on our perspective today, the device color spaces merely provided a specification for different process color models. Because the color appearance is device dependent, these spaces are actually the worst choice for faithful color reproduction across different devices.

Next, calibrated color was invented, along with the idea of color conversions through device independent color. The advantage of this method is a significant improvement in color matching across different devices. PDF follows by adding calibrated color spaces. Later on, calibrated color evolved into a standard form - the ICC profile. When color management based on ICC profiles gained its popularity and became the de-facto standard among color professionals, PDF added the ICC-based color space, which allows for embedding ICC profiles into PDF documents.

PDF Color Conversion

PDF color coversion can be described as a function of source color space, destination color space, and rendering intent. Please take note of the rendering intent values:

ColorSync/ICC Color Model

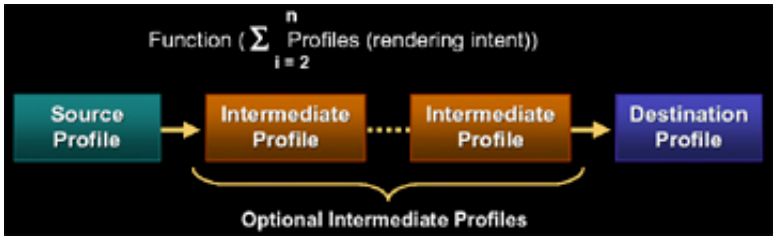
Now let's take a look at how ICC and ColorSync define color and color conversion.

ICC color definition

Naturally, color is defined by an ICC profile. As mentioned previously, the ICC profile is the most general form of the color space description.

ICC color conversions

ICC color conversions are very similar to those in PDF. The differences are an option to insert an intermediate profile between the source and destination profiles:



These additional profiles are used for soft-proofing, color device simulation, applying special affects, etc. As we can now see, the rendering intent values in ICC color conversions are the same as those in PDF:

Rendering intent = {perceptual, relative colorimetric, saturation, absolute colorimetric}

Color Model in Quartz

Integration of PDF and ICC color models

The color model in Quartz was created by integrating the ICC and PDF color models. Here're some of the interesting features of this model. All PDF color spaces are expressed in Quartz as ICC profiles. Device color spaces are assigned default profiles as follows:

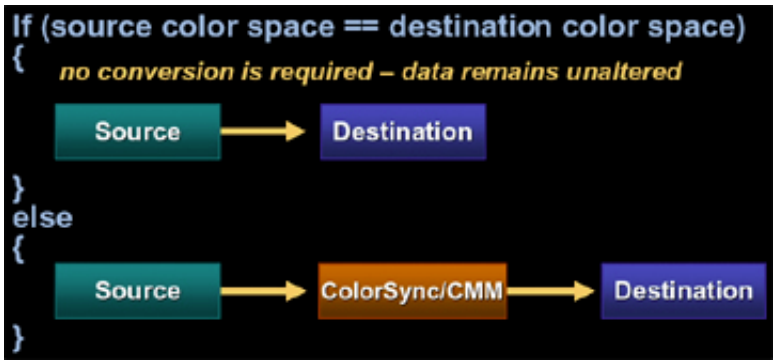
- /DeviceGray -> Quartz Default Gray
- /DeviceRGB -> Quartz Default RGB
- /DeviceCMYK -> Quartz Default CMYK

Calibrated color spaces contain the calibration record, which can be very easily re-packaged as the ICC profile. Finally, ICC-based color spaces provide their own ICC profiles.

PDF color space equivalence

Another simple but very important concept that Quartz inherited from PDF is color space equivalence. An implied rule is

that color conversions are necessary only if the source color space is different from the destination. Quartz takes advantage of this simple rule to properly organize the flow of color data through multiple rendering stages:



Color space is one or more ICC profiles

As mentioned previously, all PDF color spaces are expressed in Quartz as an ICC profile in a manner which is seamless and transparent through all applications working in the PDF imaging model. But at the same time, we needed a provision in Quartz to create color transformations for more than two profiles. For that reason, we designed a color space which can consist of one or more ICC profiles. This way we are able to preserve the PDF concept of matching a single source to a single destination, but at the same time we are able to create those complex color transformations which are suitable for advanced color management. And if the need arises for such a color space to be embedded in PDF, the multiple profiles contained in such a color space can be concatenated by ColorSync into a single profile.

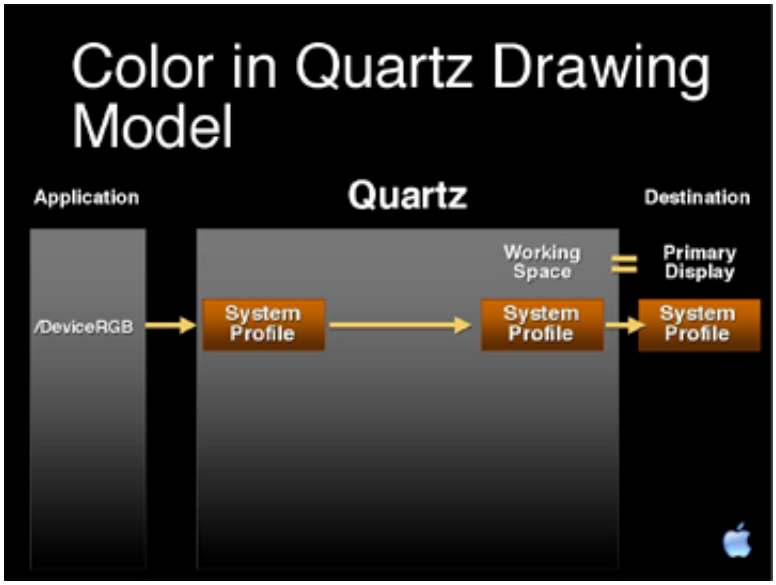
Color in Quartz Drawing Model

Let's now see how all the things just discussed come together to provide the desired color processing in the Quartz drawing model. From the top-level perspective, there are three main components involved in this model:

- An application generating the graphic contents
- Quartz providing the rendering services
- The destination for the rasterized data

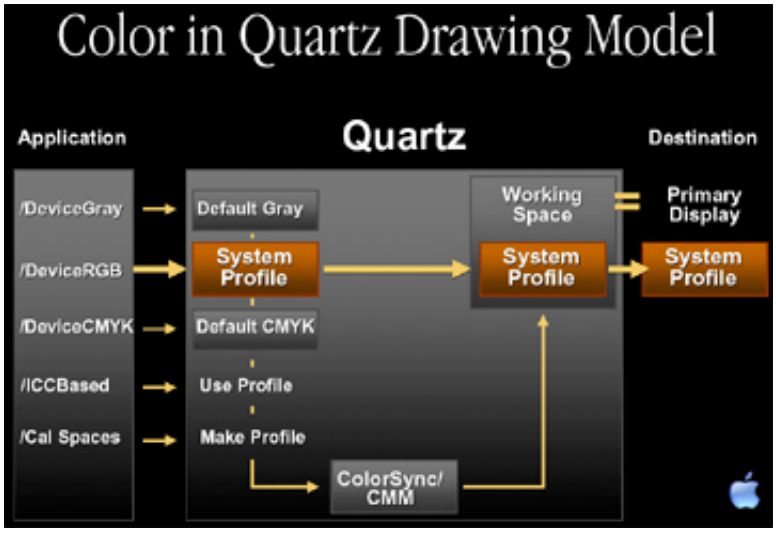
In terms of color processing, the application can request Quartz to render data in any of the PDF color spaces. To accomplish that, Quartz will convert all these into ICC profiles. Default profiles are selected by Quartz, but can be overridden by the application. Quartz will do all compositing in the working space, and ColorSync will be invoked to perform color transformations from any of the source color spaces to the working space. Finally, when compositing is completed, the rasterized data will be sent to the desired destination. Naturally, if the color space of the destination doesn't match the working space, and additional conversion must be done.

Here is an example of Quartz color management usage. An application creates content in the `/DeviceRGB` color space. After being rendered, that content should be displayed on the primary monitor without any color conversions. From the color management point of view, we can classify this case as creating and rendering the data in the same color space:



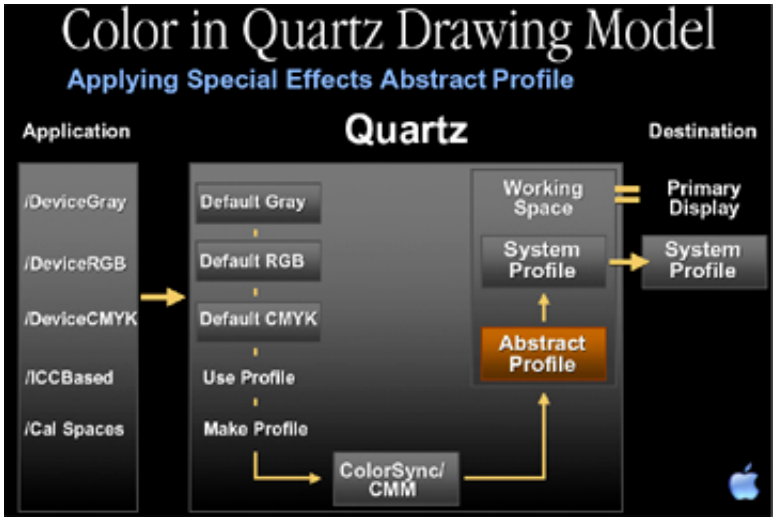
This case could be a typical example of a very simple application which doesn't have any color management capabilities. If that were the old system, and the application opened a file which contained data in a color space other than `/DeviceRGB`, then the application itself would have to convert the data to the system profile. Otherwise, displaying such data would result in color errors or might not even be possible.

Let's see what happens in Quartz if the application does not do anything about converting color and instead passes the data directly to Quartz. In this case, all conversions are done correctly in Quartz, and the data from different color spaces is converted properly to the primary display:

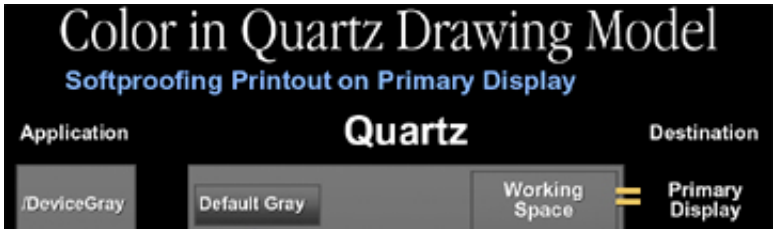


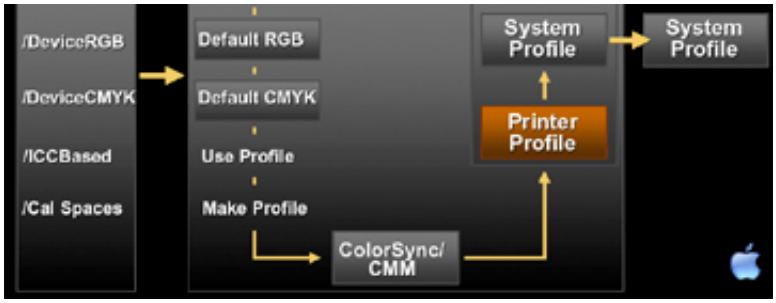
As a matter of fact, this configuration is the pre-defined default setting used by Quartz. This same configuration could be used by an advanced color application using Quartz only for rendering without any color conversion. Obviously, many other such combinations are also possible.

Let's see how convenient Quartz color management can be for some other color operations. For example, applying special effects to all data can be very easily achieved simply by adding the abstract profile to the working space. As shown in the following diagram, all conversions to the working space will include the transformation defining the abstract profile:

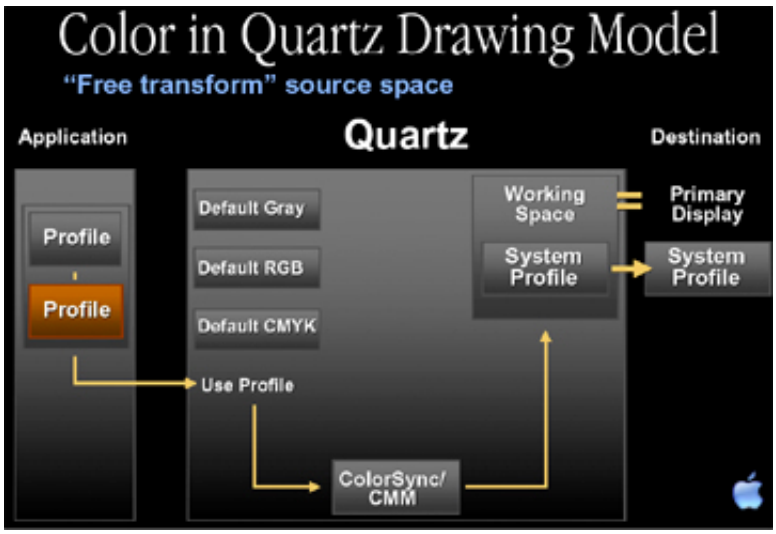


Another example could be soft-proofing. The only thing we need to do is add the printer profile to the working space. This way, all color corrections defining the printer profile will be reflected in the working space, and thus will be shown on the primary display



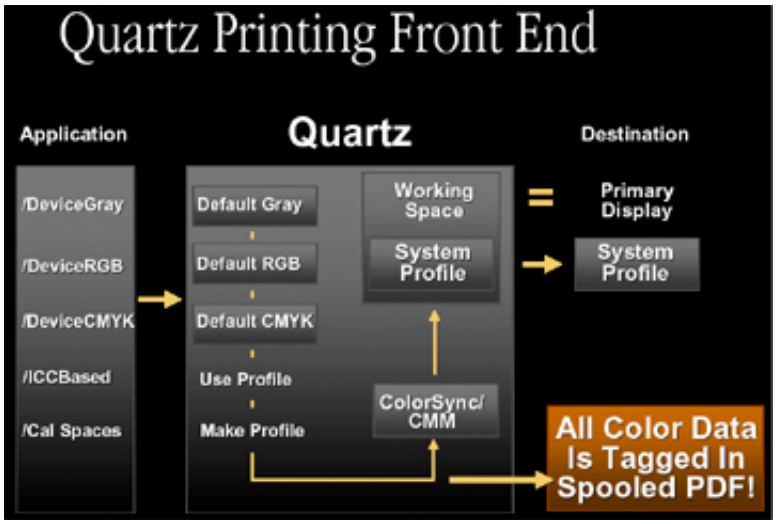


The use of multi-profile color spaces is not limited to the Quartz internals. Applications can also use them. Here is an example of an application using a given color space to create a free-transform color space to produce certain color effects.



Quartz Printing Front End

To complete our story about color management in Quartz, let's look at how color is handled in printing from Quartz. So far we've talked about the main flow of color data from the source to the destination. But one more step is also possible here. The contents can be spooled in PDF form for printing, which will be handled by the print center. There is one very important fact about the spooled data. As was pointed out, all color data is tagged in spooled PDF. And profiles are assigned to the data in exactly the same way they are for ColorSync processing:

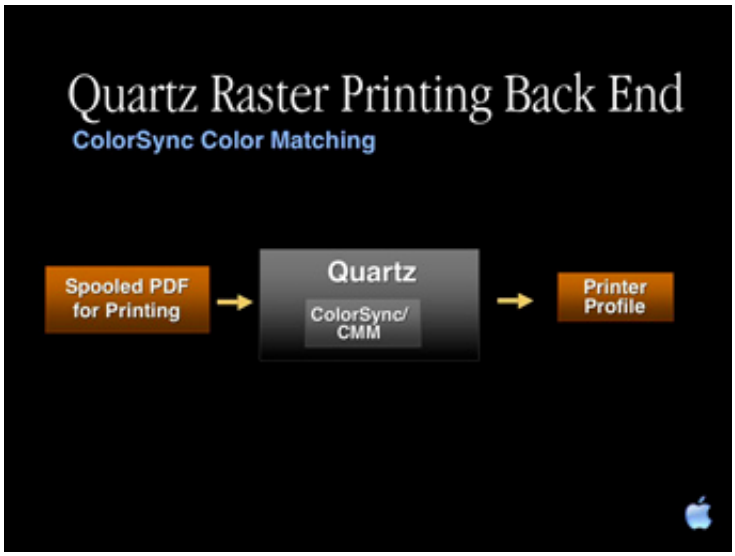


Quartz Raster Printing Back End

ColorSync Color Matching

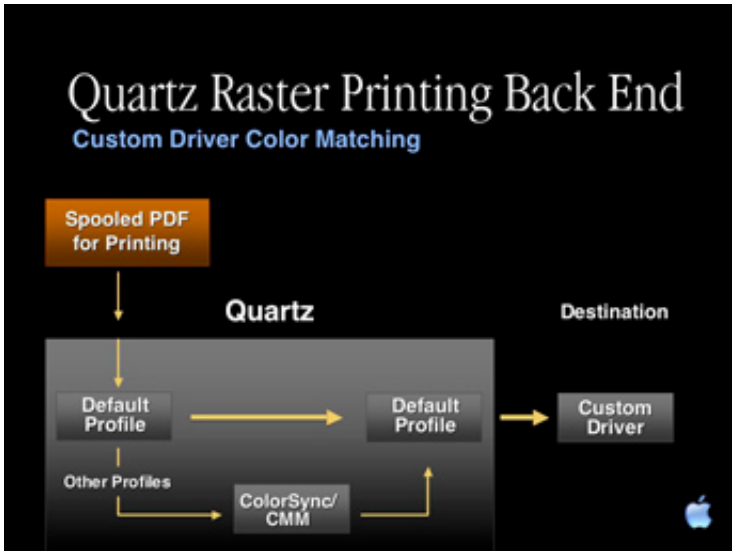
Quartz is also used to rasterize this spooled PDF at the printing back-end. From the color management perspective, we

have two options here. The first option is to use ColorSync for color matching. In this case, all color data from the spooled PDF is converted to the printer profile (Note: there is no device data in the spooled PDF).



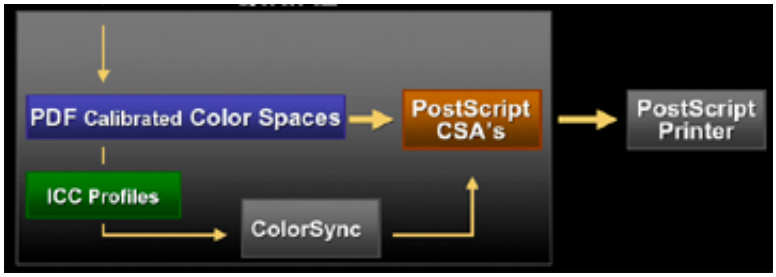
Custom Driver Color Matching

Another option is custom driver color matching.The color data, by design, is going to be handed off in the color space of the default profile provided by the system. As a result, all data which is tagged with the same default profile will remain untouched. And obviously all other color data will be converted by ColorSync to the system profile.



Quartz in PostScript Back End

ColorSync is also involved in the PostScript back end. From the color management perspective, our goal here is to convert the color spaces contained in the spooled PDF to PostScript color space arrays (CSAs). Quartz has the internal capability to convert PDF calibrated color spaces directly to PostScript CSAs. All ICC profiles will be converted to PostScript CSAs using ColorSync.



Back to top

Profiles on Mac OS X

On Mac OS 9, profiles are traditionally stored in the Profiles folder within the System folder. An application wanting to gain access to this folder would use the `CMGetColorSyncFolderSpec` API, passing in `kSystemDisk` for the `vRefNum` parameter as follows:

```
CMGetColorSyncFolderSpec(kSystemDisk, ...);
```

You could also use the Mac OS FindFolder API as an equivalent technique. However, one of the key features of Mac OS X is it is designed to be fully network and multi-user savvy. For this reason, there is no longer just one location where profiles can be stored, but several. There is a special location within the Mac OS X System folder for profiles:

/System/Library/ColorSync/Profiles/

This is where ColorSync stores profiles it uses internally for critical operations, should others be mistakenly lost or damaged. This folder is locked and protected, but if you need to get access to this folder to be able to read these profiles you can again use the `CMGetColorSyncFolderSpec` API, passing the `kSystemDomain` constant for the `vRefNum` parameter as follows:

```
CMGetColorSyncFolderSpec(kSystemDomain, ...);
```

ColorSync profiles are primary stored here:

/Library/ColorSync/Profiles/

This is where ColorSync stores the majority of its profiles. This is a read/writable folder. If you need to get access to this folder to be able to read these profiles use the `CMGetColorSyncFolderSpec` API, passing the `kLocalDomain` constant for the `vRefNum` parameter as follows:

```
CMGetColorSyncFolderSpec(kLocalDomain, ...);
```

If you are in a network environment, and your network administrator has stored profiles on the network for network devices, these can be stored in:

/Network/Library/ColorSync/Profiles/

If you need to get access to this folder to be able to read these profiles use the `CMGetColorSyncFolderSpec` API, passing the `kNetworkDomain` constant for the `vRefNum` parameter as follows:

```
CMGetColorSyncFolderSpec(kNetworkDomain, ...);
```

Lastly, users can store their own personal profiles in their home directory:

~/Library/ColorSync/Profiles/

If you need to get access to this folder to be able to read these profiles use the `CMGetColorSyncFolderSpec` API, passing the `kUserDomain` constant for the `vRefNum` parameter as follows:

```
CMGetColorSyncFolderSpec(kUserDomain, ...);
```

As on Mac OS 9, ColorSync on Mac OS X supports looking in subfolders of all the above directories (one level deep), or aliases to these folders or profiles. In summary, there are a great many locations where profiles can be stored on Mac OS X. If your application would like to present a list of profiles to the user,
we suggest you use the `CMIterateColorSyncFolder` API. This API will greatly simplify your code, as it searches all the above locations for profiles. In addition, it offers great performance benefits because it caches frequently used

information.

One important note with respect to this API - if you have a Mac OS Classic system folder setup in addition to the Mac OS X system folder, ColorSync by default will not search this Mac OS Classic system folder for profiles. However, if your application wishes to access profiles in this location, use the `CMIterateColorSyncFolder` API passing `kClassicDomain` for the `vRefNum` parameter:

```
CMGetColorSyncFolderSpec(kClassicDomain, ...);
```

In summary, an application should install profiles in:

/Library/ColorSync/Profiles/

or in a subfolder of it. Users can install personal profiles in:

~/Library/ColorSync/Profiles/

Profile prerequisites

Traditionally, profiles were filtered based on their type and creator on Mac OS 9 (type `'prof'`, creator `'sync'`). However, profiles may not have a type and creator, so ColorSync no longer checks the type and creator of profiles. Also, profiles need not have any suffix. However, if you do use one use `'.icc'`. By standardizing on one suffix it makes it easier for applications using the Cocoa `NSOpenPanel` APIs to filter profiles. Similarly, if you use a type and creator for your profile, use `'prof'` and `'sync'`, as this will make it easier for the Carbon Navigation Services APIs to filter profiles. At its heart, ColorSync no longer cares about type, creator, or suffix.

What really distinguishes a ColorSync profile from any old file is if it is a valid ICC profile. ColorSync determines if a file is a valid ICC profile by looking for the signature bytes `'acsp'` (stands for "a ColorSync profile") at an appropriate offset in its header block. ColorSync also imposes the additional restriction that the profile contain a valid description tag. Many applications will present lists of profiles to the user, and if a profile has a bad name in the description tag, it will show as garbage or not at all. In order to avoid these problems, we suggest you use the ColorSync Profile First Aid utility to make sure your profiles are free of common errors.

New optional profile tags

One of the key features of Mac OS X is it is a multi-localized operating system. ColorSync now provides this same functionality for ICC profiles. Currently, the ICC format defines a required tag `'desc'` which stores ASCII, UniCode, and ScriptCode versions of the profile description for display purposes. However, this structure allows the profile to be localized for one language only through UniCode or ScriptCode. Profile vendors have to ship many localized versions to different countries. It also creates problems when a document with localized profiles embedded in it is shipped to a system using a different language. ColorSync has defined a new optional tag to remedy this situation:

- `'mluc'` Multi-localized UniCode Tag

This tag contains a set of multilingual Unicode strings associated with a profile. We also provide a number of new APIs for easy access to this tag:

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

prof - profile to query
tag - tag type of profile to query
dict - returns dictionary

This API allows you to get a `CFDictionary` which contains the language locale and string for multiple localizations from a given tag. Similarly, there is a `CMSetProfileLocalizedStringDictionary` API to allow you to write a dictionary of localized strings to a given tag in a profile:

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

prof - profile to modify
tag - tag type of profile to modify
dict - dictionary to modify

However, most applications will simply want to get one specific string out of a profile. For this reason, we provide the `CMCopyProfileLocalizedString` function:

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

prof - profile to query
tag - tag type of profile to query
reqLocale - requested locale (optional
locale - returns locale (optional)
dict - returns dictionary string (optional)

Here's a short example showing how to use this function. We pass in the optional tag `'dscm'` plus "enUS" for the `reqLocale` parameter, looking for a U.S. Enlish string. If a U.S. English string is not found, ColorSync will fall back to a reasonable default:

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

ICC4 Profiles

The ICC has been working on a new format for profiles to provide for more flexibility and power. This new format is the ICC 4.0 profile format specification. Some of the highlights include:

- New version in header

  to distinguish from earlier versions

- New MD5 (message digest) checksum in header

  to tell if two profiles are identical , or if a profile has changed over time. You can access this new MD5 checksum directly in the profile header, or alternately there is a new ColorSync API `CMGetProfileMD5`. The ColorSync API has the advantage that it works with both ICC 4 profiles and earlier profiles.

- New tag data types (`'mluc'`, `'mBA '`, `'mAB '`, `'para'`)

These new tags really demonstrate the new power and flexibility of the ICC 4 profiles. One important note is some of your existing tags may now contain these new data formats e.g., `'A2B0'` may contain `'mft1'` or `'mAB '` data. The good news is ColorSync and the Apple CMM will be fully ICC4 savvy. This means if your application does not parse profiles, but simply uses profile references for color matching, everything will "just work". However, if your application creates of modifies profiles, you need to be aware of these new tag types to make sure you handle these cases correctly.

Back to top

CMMs on Mac OS X

As with profiles, CMMs can also be installed in multiple locations. Typically, applications would install CMMs in:

/Library/ColorSync/CMMs

Also, users may install personal CMMs (for example, for debugging) in:

~/Library/ColorSync/CMMs

If your application would like to present a list of CMMs to the user, use the `CMMIterateCMMInfo` API. See TechNote 1160 for more information. This will give you information about all the CMMs installed on the system, including the default CMM.

Building CMMs

CMMs on Mac OS X are CFBundle based, whereas on Mac OS 9 they are Component Manager based. See the CFBundle documentation for details on using these APIs to build your CMM as a CFBundle.

CMMs on Mac OS X still contain all the familiar entry points that exist for CMMs on Mac OS 9 such as `CMMOpen`, `CMMConcat`, `CMMMatchBitmap`, `CMMClose`, etc. However, Mac OS X CMMs now have a CFBundle wrapper around these entry points instead of a Component Manager wrappe

The Apple CMM has undergone a lot of work to make sure it is finely tuned and integrated fully with Quartz to handle all the various profile types and image color spaces correctly. If you are writing your own CMM, make sure and test it thoroughly, and

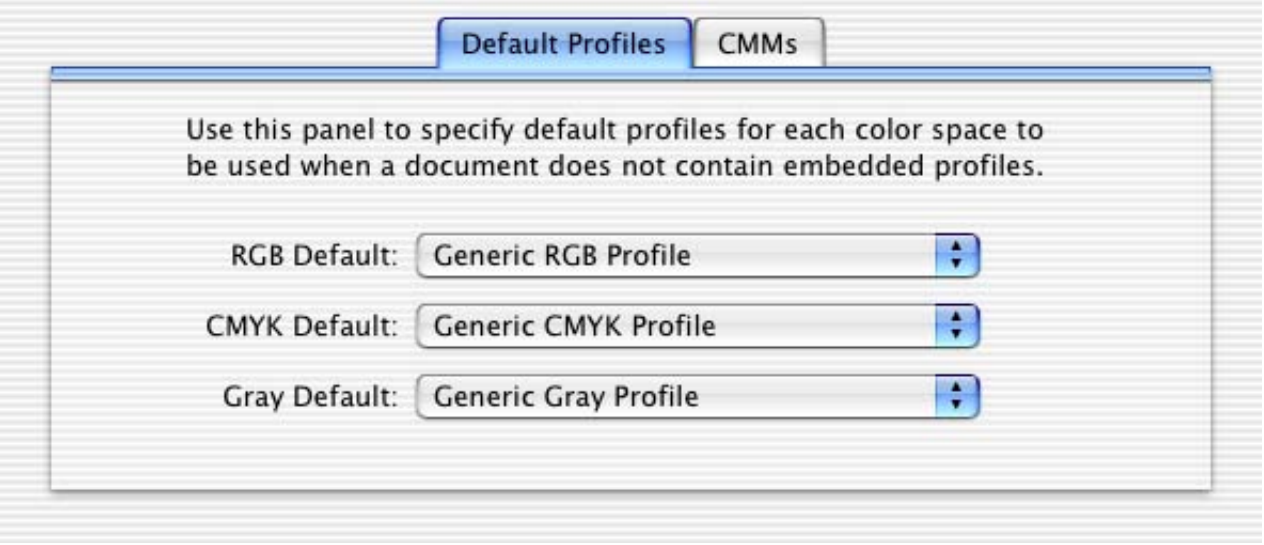
under a wide variety of cases to ensure it works in this environment.

Back to top

ColorSync Preferences and Supporting APIs

Under Classic Mac OS, applications performing color management generally do not have knowledge of the various devices on the system. Because of this, applications would find it necessary to present some kind of interface to obtain device color information from the user. Similarly, applications performing color management on data with no associated profile would need to understand what type of device the data originated from. Again, this would require the application present some kind of interface to allow the user to select a default profile for a color space or document. Over time, users were becoming confused by the great myriad of different interfaces being presented by applications performing color management. ColorSync on Mac OS X is now aware of system devices, and the ColorSync user interface has been redesigned to take advantage of this information.

The ColorSync prefences panel was redesigned in ColorSync 3.0 to provide a single point for users to go to control color via ColorSync profiles on their system. It was built on a model which is easily extensible, therefore support for new devices and color spaces may be added in the future. Here's what the new panel looks like:



There are two different panels in the control panel. The first panel is the "Default Profiles" panel. You can use this panel to specify default profiles for each color space to be used when a document does not contain embedded profiles.

Profiles for Standard Color Spaces

Documents can of course contain many different types of data. With this menu users can specify profiles for documents for different color spaces. Of course if the document does have a profile, this will take precedence, but often times documents don't have profiles associated with them. Users can retrieve this preference via the `CMGetDefaultProfileBySpace` API as follows:

```
enum {
cmXYZData = 'XYZ ',
cmLabData = 'Lab ',
cmRGBData = 'RGB ',
cmSRGBData = 'sRGB',
... };

CMGetDefaultProfileBySpace ( OSType space,
CMProfileRef* prof );
```

space - the color space
prof - returns the default profile for the specified color space

Here's sample code demonstrating how an application might benefit from these APIs. The sample accepts an image reference as an input and attempts to obtain information or data from this reference. The first thing an application should do when managing color for a document is to determine whether or not any embedded profile exists for the document. Most of the modern image formats have containers for profiles, so there are known ways to get profiles for documents in these formats. However, if there is no profile associated with the document, we can conveniently use the `CMGetDefaultProfileBySpace` API (after first obtaining

the image color space) to get a reasonable default source profile for the document.

Next, we obtain a destination profile for the document. We can use the CMGetDefaultProfileByUse to get a destination profile. With both a source and destination profile we are now able to perform our color management - in this example, we use NCWNewColorWorld and CWMatchBitMap.

One important note regarding this example is the code shown here is appropriate for a Mac OS Classic or Carbon application running on Mac OS 9. On Mac OS X, much of the destination color management is done for you, so you don't need this matching if your application uses Quartz. However, you need to at least be aware of where the data is coming from, and if no profile is associated with it you can tag it using the ColorSync APIs.

ColorSync Prefs Code Sample

```
CMError myMatchImage (myImageRef image)
{
    CMProfileRef source, dest;
    CMWorldRef cw;
    CMBitMap bm;

    source = myGetEmbeddedProfile (image);
    if (source == nil)
    {
        space = myGetColorImageSpace (image);
        err = CMGetDefaultProfileBySpace (space, &source);
    }

    err = CMGetDefaultProfileByUse (cmDisplayUse, &dest);
    err = NCWNewColorWorld (&cw, source, dest);
    bm = myGetImageBitMap (image);
    err = CWMatchBitMap (cw, &bm, ..., ..., ...);
...
```

ColorSync Preferences Tidbits

As with ColorSync 3.0 on Mac OS 9, ColorSync applications on Mac OS X can also launch the ColorSync control panel to solicit color choices, including default profiles for devices and documents, as well as the preferred CMM. Users can also switch between named collections of color settings called workflows.

To launch the ColorSync control panel from your application, simply call the following function:

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

flags - You must pass a value of 0 for this parameter. Future versions of ColorSync may define constants for the flags parameter to specify how the ColorSync control panel is launched.

function result - A result code of type CMError.

When your application calls the CMLaunchControlPanel routine, any changes made by the user will not be available (through calls such as CMGetDefaultProfileBySpace) until the user closes the ColorSync control panel. There is currently no ColorSync API for determining if the ColorSync control panel has been closed, though on Mac OS 9 you can use the Process Manager API for this purpose.

ColorSync Preferences Summary

Since all data is color managed by Quartz on Mac OS X, your application can participate simply by making sure your source and destination data is associated with a profile. Most applications can use the ColorSync Preferences APIs for color management, to get user's preferences. And these profile accessor APIs are extensible, a bridge to future device and color-space support. Lastly, the ColorSync Preferences APIs offer great flexibility in that applications can manage color from both document-centric and device-centric perspectives.

Back to top

ColorSync Device Support

Mac OS X contains new device managers for input, display, and printing. This provides an opportunity for developers to integrate

with ColorSync and provide both awareness of devices and access to their profiles. This new device support has been implemented with the following services:

- Device registration
- Profile registration
- Default Device and Default Profile accessors
- Notification

Device and Profile Registration:

ColorSync relies on the Device Managers to detect the presence of devices and to discover or build device profiles. The following are new ColorSync APIs which you can use to capture device and profile information:

```
CMRegisterColorDevice

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

`deviceClass`	The class of the device (e.g., `'scnr'`, `'cmra'`, `'prtr'`, `'mntr'`)
`deviceID`	Unique identifier per class (Class + ID uniquely id's device)
`deviceName`	Name of the device.
`deviceScope`	Structure defining the user and host scope this device pertains to.

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

CMUnregisterColorDevice

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

`deviceClass`	The class of the device (e.g., `'scnr'`, `'cmra'`, `'prtr'`, `'mntr'`)
`deviceID`	Unique identifier per class (Class + ID uniquely id's device)

When a device is no longer to be used on a system (as opposed to being offline), it should be unregistered. If a device is temporarily shut down or disconnected, it need not be un-registered unless the device driver: a) "knows" that it will not be used (being de-installed) b) cannot access the device profiles without the device. If either of these are true, the device should be un-registered.

CMSetFactoryDeviceProfiles

```
CMError CMSetFactoryDeviceProfiles
(
 CMDeviceClass        deviceClass,
 CMDeviceID        deviceID,
 CMDeviceProfileID    defaultID,
 const CMDeviceProfileArray*    deviceProfiles
);
```

Parameters

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

deviceID	Unique identifier per class (Class + ID uniquely id's device)
defaultID	The id of the default profile for this device.
deviceProfiles	Ptr to caller's storage containing the profile array.

This API establishes the profiles used by a given device. It should be called after device registration to notify ColorSync of the device's profiles. Note that a distinction is made in the API between the "factory" device profiles and the current device profiles, since the latter may contain modifications to the factory set.

Default Devices and Default Profiles:

These new APIs for profiles and standard devices provide access to defaults, not just any device profile. When profiles are registered for a device, one is identified as the default for that device. However, over time, the user may change their settings. When a user changes settings, the default device or default profile may change, and ColorSync is made aware of such changes by the Device Managers. The Device Managers keep track of which device is default.

Here are some new APIs to manage default devices and profiles:

CMSetDefaultDevice

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

Parameters

Name	Description
deviceClass	The class of the device (e.g., 'scnr', 'cmra', 'prtr', 'mntr')
deviceID	Unique identifier per class (Class + ID uniquely id's device)

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

CMSetDeviceDefaultProfileID

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

The default profile ID for a given device is an important piece of information because of the CMGetProfileByUse API. This API will return the default profile for devices depending on the user's selection in the ColorSync Control Panel. Device drivers and host software can get and set the default profile for a given device with this API.

Parameters

Name	Description
deviceClass	The class of the device (e.g., 'scnr', 'cmra', 'prtr', 'mntr')
deviceID	Unique identifier per class (Class + ID uniquely id's device)
defaultID	The id of the default profile for this device.

Calibration Support:

These new Device Managers on Mac OS X now make if possible for ColorSync to provide support for calibration. This is made possible by the data ColorSync is given for a profile by the Device Managers. Profiles are registered with an ID and a "mode" name (e.g., "plain paper"). Profiles are referenced during processing by ID. Such profiles can be customized by an application. For example a calibration application can get the factory profile for a given calibration mode, and then set a new profile (by ID) to be used for that mode.

Here are some new APIs supporting profile customization:

CMGetDeviceFactoryProfiles

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

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

Parameters

Name	Description
`deviceClass`	The class of the device (e.g., `'scnr'`, `'cmra'`, `'prtr'`, `'mntr'`)
`deviceID`	Unique identifier per class (Class + ID uniquely id's device)
`defaultID`	Ptr to storage for the id of the default profile for this device.
`arraySize`	Ptr to storage for the size of the array to be returned. The caller may first call this routine to get the size returned, then call it again with the size of the buffer to receive the array.
deviceProfiles	Ptr to callers storage to receive the profile array. The caller may first pass NIL in this parameter to receive the size of the array in the `arraySize` parameter. Then, once the appropriate amount of storage has been allocated, a pointer to it can be passed in this parameter to have the array copied to that storage.

CMSetDeviceProfiles

```
CMError
CMSetDeviceProfiles(
  CMDeviceClass                 deviceClass,
  CMDeviceID                    deviceID,
  const CMDeviceProfileScope *  profileScope,
  const CMDeviceProfileArray *  deviceProfiles);
```

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

Parameters

Name	Description
`deviceClass`	The class of the device (e.g., `'scnr'`, `'cmra'`, `'prtr'`, `'mntr'`)
`deviceID`	Unique identifier per class (Class + ID uniquely id's device)
`profileScope`	Structure defining the scope these profiles pertain to.
`deviceProfiles`	Ptr to caller's storage containing the profile array which contains replacements for the factory profiles. Not all the original profiles must be replaced with this call. Therefore the array can contain as few as 1 profile and as many as there were in the factory array if they are all to be replaced. Profiles are repaced by ID.

```
CMSetDeviceProfile

CMSetDeviceProfile


CMError
CMSetDeviceProfile(
```

```
CMDeviceClass                deviceClass,
CMDeviceID                   deviceID,
const CMDeviceProfileScope *  profileScope,
CMDeviceProfileID            profileID,
const CMProfileLocation *    deviceProfLoc);
```

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

Parameters

Name	Description
`deviceClass`	The class of the device (e.g., `'scnr'`, `'cmra'`, `'prtr'`, `'mntr'`)
`deviceID`	Unique identifier per class (Class + ID uniquely id's device)
`profileScope`	Structure defining the scope these profiles pertain to.
`profileID`	The id of the default profile for this device.
`deviceProfLoc`	Ptr to storage for the `CMProfileLocation` of the profile. Since this structure is a fixed length structure, the caller can simply pass a ptr to a stack-based structure or memory allocated for it.

Notifications

Applications now can be made aware of changes to devices on their system. ColorSync will post notifications to a distributed notification center (check the Cocoa and Core Foundation documentation for additional details on distributed notification centers). Any process on the system which registers with a distributed center will receive these notifications. Here's some of the available notifications:

- Changes to the default device for a device class
- Changes to a device's factory or custom profiles
- Changes to a device's default profile
- Device registration/un-registration

There are no ColorSync APIs to register to receive the above notifications - ColorSync only posts these notifications to the distributed center. Instead, use the following Cocoa and Core Foundation APIs to receive the above notifications:

```
CFNotificationCenterAddObserver
NSDistributedNotificationCenter
```

Here's the specific notification strings (from the ColorSync interface file `CMDeviceIntegration.h`) which you can use with the above functions:

```
#define  kCMDeviceRegisteredNotification   CFSTR("CMDeviceRegisteredNotification")
#define  kCMDeviceUnregisteredNotification  CFSTR("CMDeviceUnregisteredNotification")
#define  kCMDeviceOnlineNotification  CFSTR("CMDeviceOnlineNotification")
#define  kCMDeviceOfflineNotification  CFSTR("CMDeviceOfflineNotification")
#define  kCMDeviceStateNotification   CFSTR("CMDeviceStateNotification")
#define  kCMDefaultDeviceNotification  CFSTR("CMDefaultDeviceNotification")
#define  kCMDeviceProfilesNotification  CFSTR("CMDeviceProfilesNotification")
#define  kCMDefaultDeviceProfileNotification  CFSTR("CMDefaultDeviceProfileNotification")
```

ColorSync Device Support Summary

ColorSync is now integrated with Device Managers. Applications have access to the default profiles for standard devices (input, display, output) via the new ColorSync preferences APIs. Applications can get more specific profile information for registered devices from these APIs. Calibration is now supported in ColorSync plus new notification services are now provided.

Back to top

ColorSync Changes for Mac OS 10.2

ColorSync for Mac OS 10.2 contains a number of new changes. These are discussed below.

Updated Profiles

Some of the ColorSync default profiles for Mac OS 10.2 have been updated. These profiles are important because they are used for any images which do not have profiles embedded in them. Here are the new changes:

- Generic RGB Profile

- was: approx. P22 phosphors, 9300°K white point, 1.8 gamma

- now: P22 phosphors, D65 white point, 1.8 gamma

- Generic CMYK Profile

- was: based on the Apple Color LaserWriter

- now: based on ANSI CGATS TR 001 data set

Profile Changes when Printing or Displaying Untagged Data

PDF Display

On Mac OS 10.2, untagged RGB data in PDF will be tagged with the Generic RGB profile, and as a result it will be color-matched to the screen.

On earlier versions of Mac OS X, untagged RGB data in PDF would be tagged with the system profile (the profile returned by the CMGetSystemProfile function), and as a result, no color-matching to the screen would occur (because both the source and destination profiles for the match would be identical).

Printing Front-End

When passing source data to the print system, the printing front-end spools the print job in the PDF format, and as described in the section above Quartz Printing Front End, all color data is tagged in the spooled PDF. Therefore, when passing untagged RGB source data to the print system for printing on Mac OS 10.2, ColorSync will tag the data with the Generic RGB profile (just as is done when displaying PDF data).

ColorSync on earlier versions of Mac OS X would instead tag the data with the system profile.

Printing Back-End

When printing on Mac OS X, ColorSync will match source data to whatever profile the printer driver provides. For this reason, printer drivers should register profiles for their devices using the Device APIs as discussed earlier in this note.

Here are some profile changes which were made in Mac OS 10.2 by the printing system back-end for both raster and PostScript printing:

Raster Printing

If a printer driver provides no profile, ColorSync on Mac OS 10.2 will substitute the Generic RGB profile. ColorSync on earlier versions of Mac OS X would substitute the system profile.

Also, in earlier versions of Mac OS X, RGB ROMM was used as an intermediate color space by the raster back-end when the printer driver provided a profile for the printer. On Mac OS 10.2 the RGB ROMM space has been removed altogether, resulting in cleaner, more accurate color.

PostScript Printing

On earlier versions of Mac OS X, originally untagged RGB data was ultimately converted to a CIEBasedABC color space based on the system profile. In Mac OS 10.2, this CIEBasedABC color space is now based on the Generic RGB profile.

Also, because all tagged color data from the spooled PDF is converted to calibrated color spaces in PostScript, on Mac OS 10.2 the printing system takes advantage of PostScript Level 2 in-RIP color matching in the printer.

Frameworks

A number of minor changes have been made to the ColorSync framework. These are described below.

ColorSync Preferences Notifications

A new device notification kCMPrefsChangeDeviceNotification has been added:

-kCMPrefsChangeDeviceNotification

This will be sent whenever a user changes a setting in the ColorSync Preferences panel (for example, if the user changes the preferred CMM).

To demonstrate how the general notification mechanism works, here's a code snippet written in Cocoa showing how you can get notified when the display profile changes. First, register with the default distributed notification center to receive the necessary ColorSync device notifications. Then, when the notification is received, simply call CMGetDefaultProfileByUse to determine the current setting for the display profile:

```
-(void)registerNotifications
{
    NSDistributedNotificationCenter *center;

    center = [NSDistributedNotificationCenter defaultCenter];

    [center addObserver:self
               selector:@selector(notification:)
                   name:kCMDeviceUnregisteredNotification
                 object:nil];
    [center addObserver:self
               selector:@selector(notification:)
                   name:kCMDefaultDeviceNotification
                 object:nil];
    [center addObserver:self
               selector:@selector(notification:)
                   name:kCMDeviceProfilesNotification
                 object:nil];
    [center addObserver:self
               selector:@selector(notification:)
                   name:kCMDefaultDeviceProfileNotification
                 object:nil];
}
-(void)notification:(NSNotification *)note
{
   CMError err = CMGetDefaultProfileByUse(cmDisplayUse, &gProfRef);
}
```

Similar code to check for profile changes could be written using the Mac OS X Core Foundation distributed notifications (using NSDistributedNotificationCenter, etc.). Check the Mac OS X Core Foundation documentation for additional information.

ColorSync Preference Pane

The "Device Profiles" tab has been removed. This tab was a legacy feature which originated from earlier ColorSync attempts at device support. Since then we've added the full device integration layer, and for this reason the old tab has been removed. Instead , this functionality can be found either in the various other device preferences, such as the Displays preference, or in the ColorSync Utility application.

The workflow import/export feature has also been removed, as it was discovered few people were actually making use of workflows. Also, because Mac OS X is a fully multi-user operating system with per-user preferences, there is no longer as great a need for workflows.

ColorSync Utility

The ColorSync Utility application has been greatly enhanced. Here's a list of the changes:

-Profile First Aid

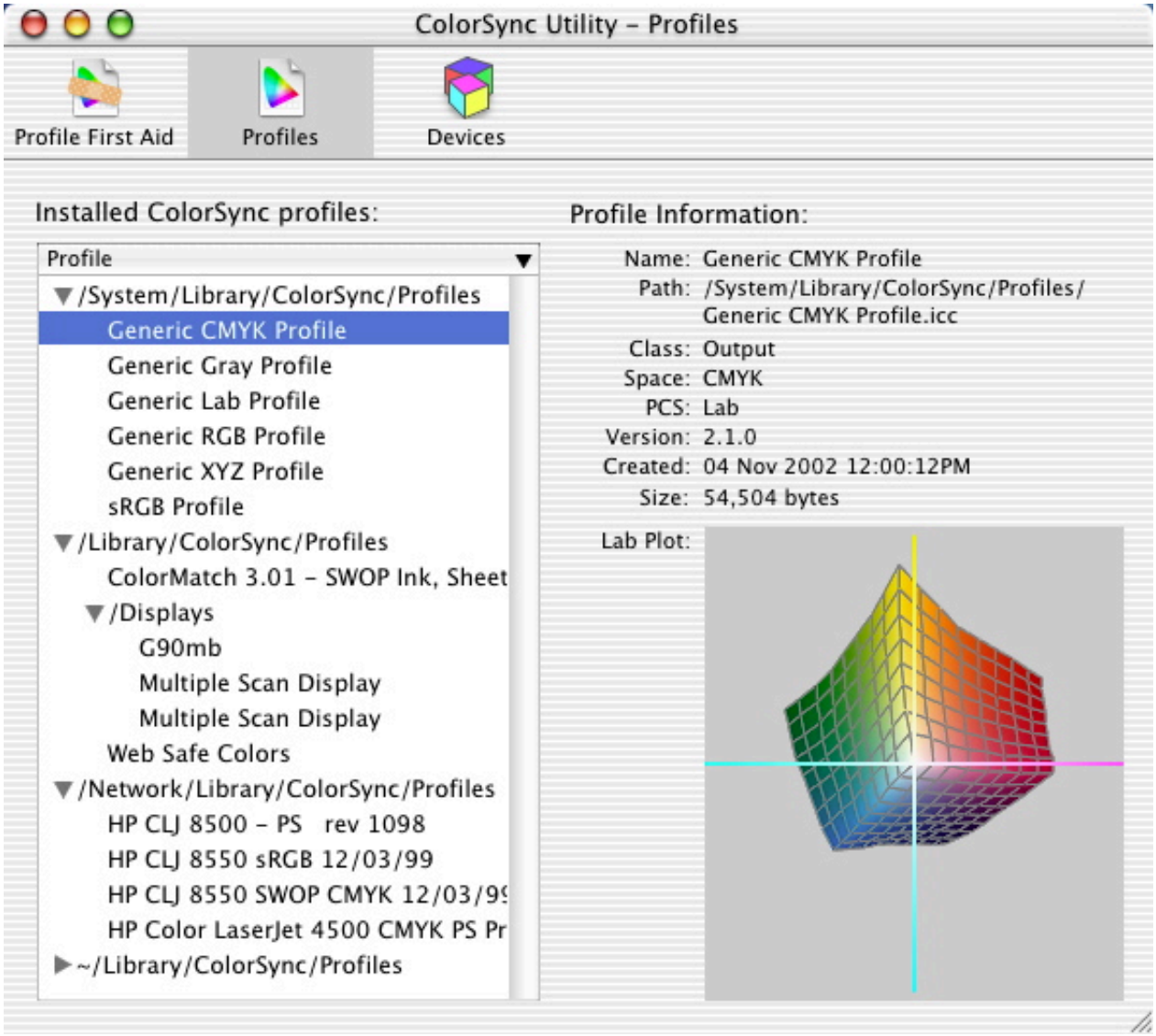
Profile First Aid allows you to diagnose and repair common problems in profiles. We've updated it for Mac OS 10.2 to help you better identify such problems. If your application or printer driver provides custom profiles, make sure and check them with Profile First Aid before you ship them to your customers.

-Devices Browser

Now that the "Device Profiles" tab has been removed from the ColorSync preferences, it's now more important than ever to be aware the devices browser is part of ColorSync Utility. This allows you to see a list of all the ColorSync devices, along with their current & factory profiles.

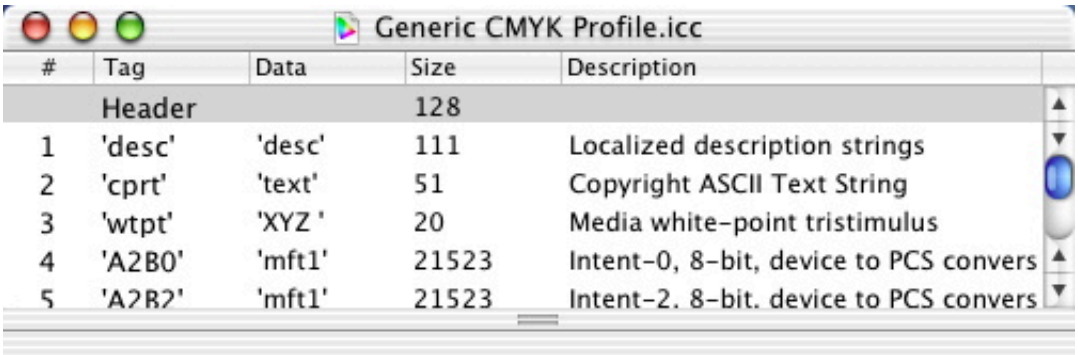
-Profile Inspection

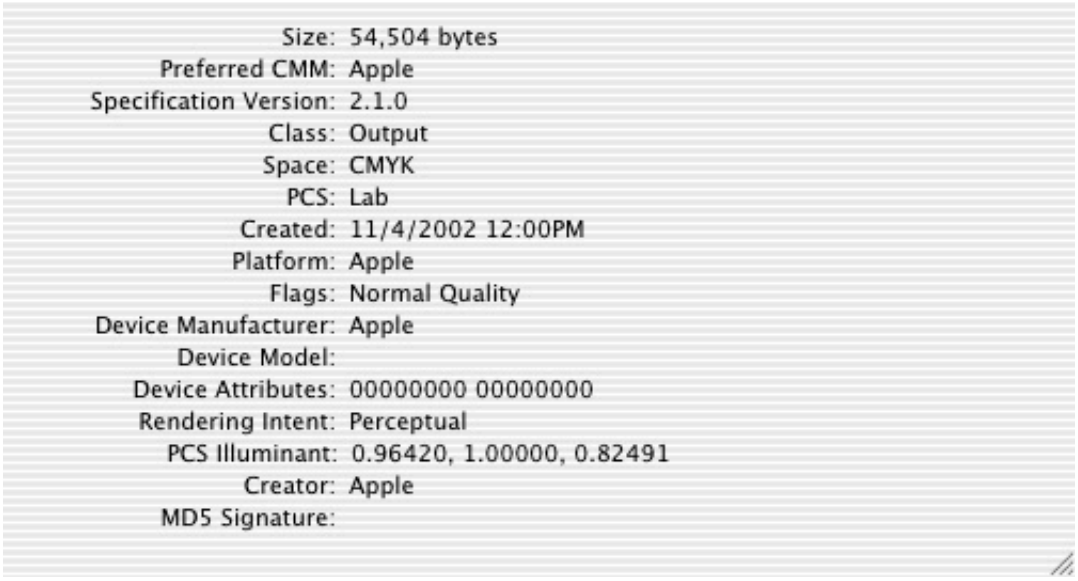
It's now possible to inspect profiles in ColorSync Utility. Simply select the profile in ColorSync Utility and you will see a 3D plot of the profile:



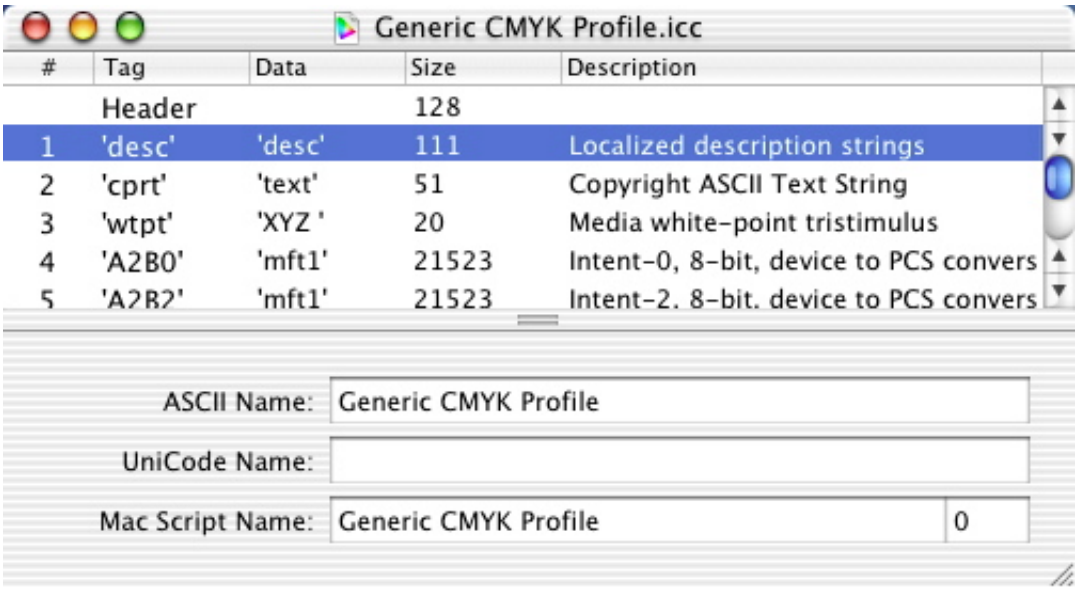
The 3D plot can be rotated. Just click and drag the plot to see how this works.

Double-click on a profile and you'll see a new window showing the profile header information and all the profile tags:

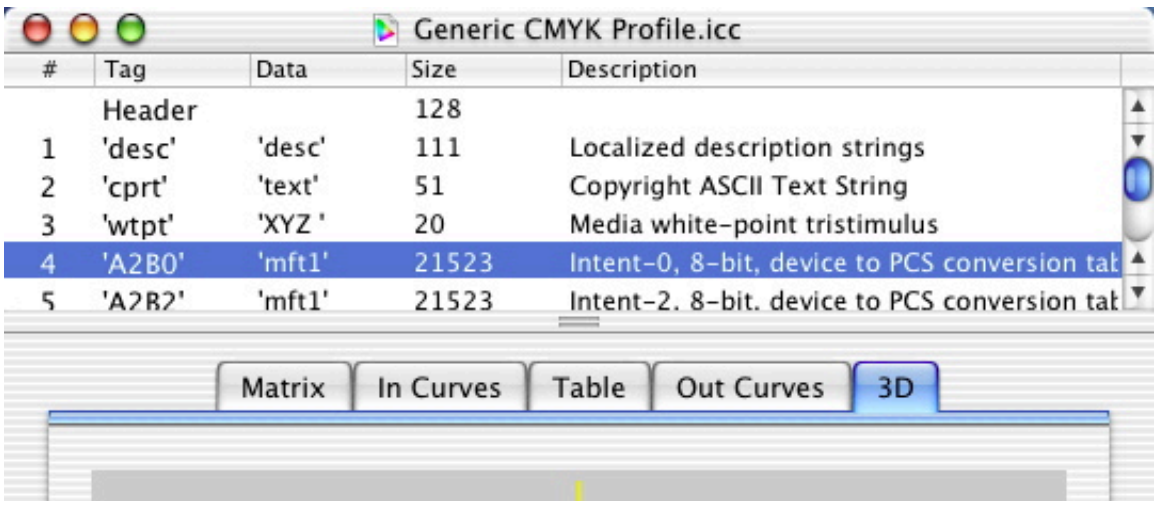


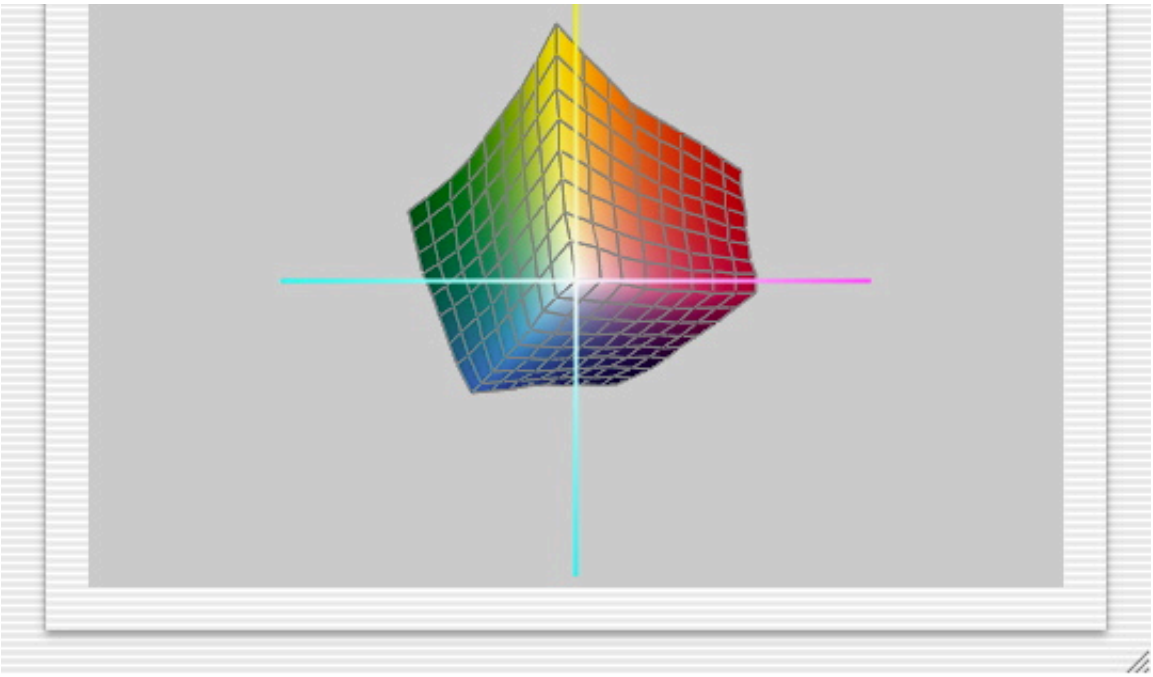


Click on a tag entry and you'll get a detailed description of the tag. For example, here's a view of the description tag, showing the ASCII name, UniCode name, and Mac Script name. These fields can be edited as well:



There's custom viewer windows for all the various profile tags. Here's one for the A2B tag. This viewer window will show the tag contents which can be navigated via tab panes. These tab panes are shown in the order in which they are applied:





Back to top

References

Inside Macintosh: Managing Color With ColorSync New ColorSync 3.0 APIs

Back to top

Downloadables

Acrobat version of this Note (76K) Download

Back to top

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code