Profiles 2

Profiles (also called color profiles) provide the KCMS framework with information on how to convert input color data to the appropriate color-corrected output color data. They are the focus of your programming efforts. A typical application loads profiles, reads profile attributes, connects profiles, optimizes profiles, and applies profiles to color data. You will probably combine or connect existing profiles to create profiles, rather than generate new ones.

Profiles include the following information:

- Color spaces in which the input and output data appears (for example, RGB, CMYK, or XYZ).
- Specific parameters of the color spaces (for example, the chromaticities of the primary colors and the tables to correct for the response of each channel).
- Specific conditions in which the colors are expected to be viewed (for example, the lighting conditions and type of media that will be used).
- Tables of data or parameters of equations that a CMM uses to transform color data. Each profile is owned by a specific CMM. Although all profiles have common, public information, some of an individual profile's format can be CMM-specific. (You do not need to understand the profile file format to write applications.)

2

KCMS, by default, uses the International Color Consortium (ICC) profile format. The ICC format is an emerging default defacto standard supported by a wide range of computer and color device vendors. This is extremely advantageous for users, as a single profile will work over multiple platforms.

Note – The ICC format is currently endorsed by the following companies: Adobe Systems Inc., Agfa-Gevaert N.V., Apple Computer Inc., Eastman Kodak Company, FOGRA (Honorary), Microsoft Corporation, Silicon Graphics, Inc., Sun Microsystems Inc., and Taligent Inc.

Profile Format

The KCMS framework uses the ICC format as the default profile format. For details on the ICC profile format, see the ICC specification. By default, it is located online in the `SUNWsdk/kcms/doc` directory.

There are some terminology differences between ICC profiles and the KCMS framework. These differences are mostly historical. Table 2-1 lists the equivalent ICC name if it is different.

Table 2-1 KCMS and ICC Profile Format Equivalents

KCMS Profile Format	ICC Equivalent
Device Color Profile	Any Input, Display, or Output Profile
Color Space Profile	Color Space Conversion Profile
Effects Color Profile	Abstract Profile
Complete Color Profile	Device Link Profile

Each color profile is owned by, or associated with a specific CMM. In general, you do not need to know which CMM owns the profile. In the case where a profile's CMM is not present and the profile is a valid ICC profile, the default CMM will provide the functionality necessary to use that profile. However, each CMM does have different ways of performing its color-correction technology. For example, each CMM has a unique way to calibrate its profiles. In addition, you may occasionally receive error codes that are CMM-specific.

For more information on CMMs, see the DDK document, *KCMS CMM Developer's Guide*.

Kinds of Profiles

The KCMS framework supports several kinds of color profiles.

Device Color Profile

A *Device Color Profile* (DCP) represents the behavior of a specific digital color device, such as a flatbed or film scanner, a computer monitor, or a printer. Each DCP specifies device color appearance under a specific set of conditions (for example, lighting type, media type, and so on). Because device behavior tends to change over time, calibration software may adjust a DCP whenever its device is calibrated. *Calibration* refers to fine tuning a specific device's color response. Typically it changes the profile data so that it can be color managed to produce the same color response as other devices of the same make and model. In other cases, depending on the device's method of calibration, the device itself is changed to match the profile.

The ICC specification separates DCPs into three categories: input, output and display. This separation can be confusing when a device, such as a printer includes input device data. The data can be considered an input profile, an output profile, or both. This occurs in print simulation where the printer is an input device to a display or other output device.

Conceptually, it may be easier to separate profiles into these three categories only in terms of how data can and cannot be sent from and to the *profile connection space* (PCS). The PCS is the common junction where profiles are connected together.

KCMS does not make this syntactical separation. Rather it considers all input, output, and display profiles as device profiles and makes no assumptions about what profiles can and cannot be connected together. The connection of the profiles is then evaluated at connection time based on the data contained within the profile.

Color Space Profile

A *Color Space Profile* (CSP) defines a color space. Colors are defined in terms directly related to spectral response. A CSP does not depend on the behavior of a particular color device. CSPs contain information about assumed viewing conditions in the data expressed for that color space. Typically, the color space

can be relative to CIEXYZ values, defined by the Commission Internationale de l'Eclairage (CIE). The equivalent ICC term for Color Space Profile is *Color Space Conversion Profile*. (See Table 2-1.)

Effects Color Profile

An *Effects Color Profile* (ECP) represents a condition that changes the appearance of colors, such as a specific kind of lighting or a simulated anomalous color vision (color blindness). In addition, an ECP can be applied for artistic purposes, such as making colors appear lighter or darker. The equivalent ICC term for Effects Color Profile is *Abstract* profile. (See Table 2-1.)

Complete Color Profile

The preceding three profile types do not contain enough information for the KCMS framework to convert color data from one form to another. Useful color transformations can only happen when your application uses the KCMS API to connect two or more profiles together to form a *Complete Color Profile* (CCP). A CCP is a connected sequence of profiles with a DCP or a CSP at either end, and possibly one or more ECPs or DCPs in between. The equivalent ICC term for Complete Color Profile is D*evice Link Profile*. (See Table 2-1.)

Getting and Setting Profile Attributes

The KCMS API provides a way to get profile information by examining the profile's *attribute set*. Each attribute has a value, which is data associated with the attribute. The C API provides the following attribute calls:

- `KcsGetAttribute()`—gets a specific attribute value associated with a profile. See "KcsGetAttribute()" on page 69 for detailed information.
- `KcsSetAttribute()`—modifies an attribute. (This is not always possible because some attributes are read-only.) See "KcsSetAttribute()" on page 86 for detailed information.

For more information on profile attributes, see Chapter 5, "KCMS Profile Attributes and ICC Tags."

Loading and Saving Profiles

Profiles are typically stored as files on disks, although they can be imbedded in an image located across a network or in read-only memory in a printer.

Profiles are loaded with the `KcsLoadProfile()` function (see page 74) and are saved with the `KcsSaveProfile()` function (see page 83). `KcsLoadProfile()` takes the three arguments listed below. `KcsSaveProfile()` takes the first two arguments listed.

- A profile identifier
- A profile description
- Hints about loading the profile

The *profile identifier* is returned to the calling program from `KcsLoadProfile()` for use with other API functions. In the case of `KcsSaveProfile()`, the identifier is passed back into the KCMS framework library to indicate the profile to be saved.

The *profile description* is a union of many different types, each of which represents a way to supply a location where the profile data should be stored. The `type` and the associated fields in the union are required to complete a profile description. The `type` field indicates which of the union's fields to use.

A calling program can request that the KCMS framework load only specific parts of a profile, (for example, just its attributes). The caller uses the `KcsModifyLoadHints()` function to provide these *load hints*, which change the load status of the profile. Hints are described by the `KcsLoadHints` data type discussed on page 40. Load hints that request specific operations and specific content be loaded for a profile are described in "Operation Hints" on page 18.

2

Using Profiles to Convert Color Data

The example in Figure 2-1 shows how color data is converted between a scanner device and a monitor device.

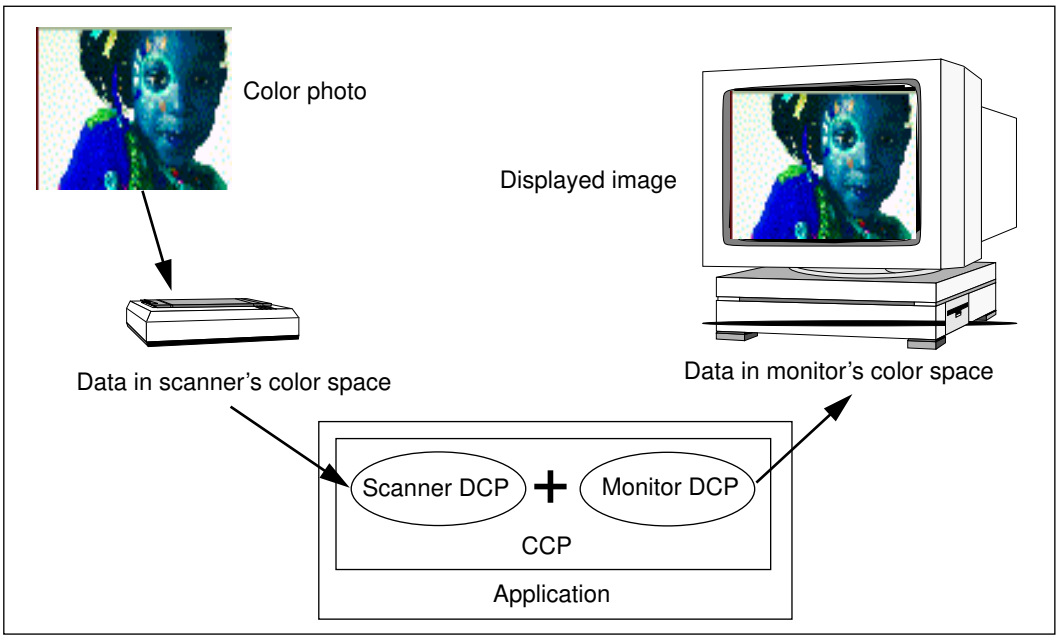

Figure 2-1 Converting Color Data From a Scanner to a Monitor

In this example, these devices do not perform their own color correction; therefore, the color data must be converted from the form provided by the scanner (Scanner DCP) to a form appropriate for display on the monitor (Monitor DCP). Follow these steps to convert the color data:

1. Load scanner and monitor profiles.
   See "Loading Scanner and Monitor Profiles" on page 13."
2. Connect a scanner profile to amonitor profile to get a complete profile.
   See "Connecting Scanner to Monitor Profiles" on page 14.
3. Evaluate color data through the complete profile.
   See "Evaluating Color Data Through the Complete Profile" on page 15.

The sequence of calls that perform this conversion is shown in Code Example 2-1. For more information on the `KcsConnectProfiles()` function, see "Using Color Space Profiles" on page 16 and the detailed function description on page 61.

Code Example 2-1 Simple Color Data Conversion

```
/* Load the scanner's DCP.*/
KcsLoadProfile(&inProfile, &scannerDescription, KcsLoadAllNow);

/* Load the monitor's DCP. */
KcsLoadProfile(&outProfile, &monitorDescription, KcsLoadAllNow);

/* Connect two DCPs to form a CCP */
profileSequence[0] = inProfile;
profileSequence[1] = outProfile;
KcsConnectProfiles(&completeProfile, 2, profileSequence,
       KcsLoadAllNow, &failedProfileIndex);

/* Apply the CCP to input color data. */
KcsEvaluate(completeProfile, KcsOperationForward, &inbufLayout,
       &outbufLayout);
```

Loading Scanner and Monitor Profiles

The `KcsLoadProfile()` function loads the profile associated with a specific device, effect, partial or complete profile. It allocates any system resources required by the profile. For a detailed description of the `KcsLoadProfile()` function, see page 74.

Connecting Scanner to Monitor Profiles

As shown in Code Example 2-1 and Figure 2-2, the next stage is to connect a pair of DCPs to form a CCP. `KcsConnectProfiles()` provides this functionality. Continuing with the example illustrated in Figure 2-1, a CCP is built by connecting the scanner's DCP to the monitor's DCP. The resulting CCP converts scanner data to monitor data.

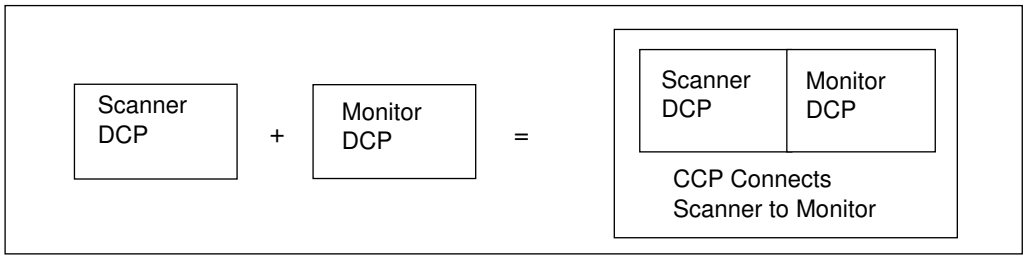

Figure 2-2 Building a CCP From Two DCPs

Evaluating Color Data Through the Complete Profile

Next the `KcsEvaluate()` function is used to apply a color transformation based on the supplied CCP. One of the following operations is associated with the evaluation. These operations are illustrated in Figure 2-3.

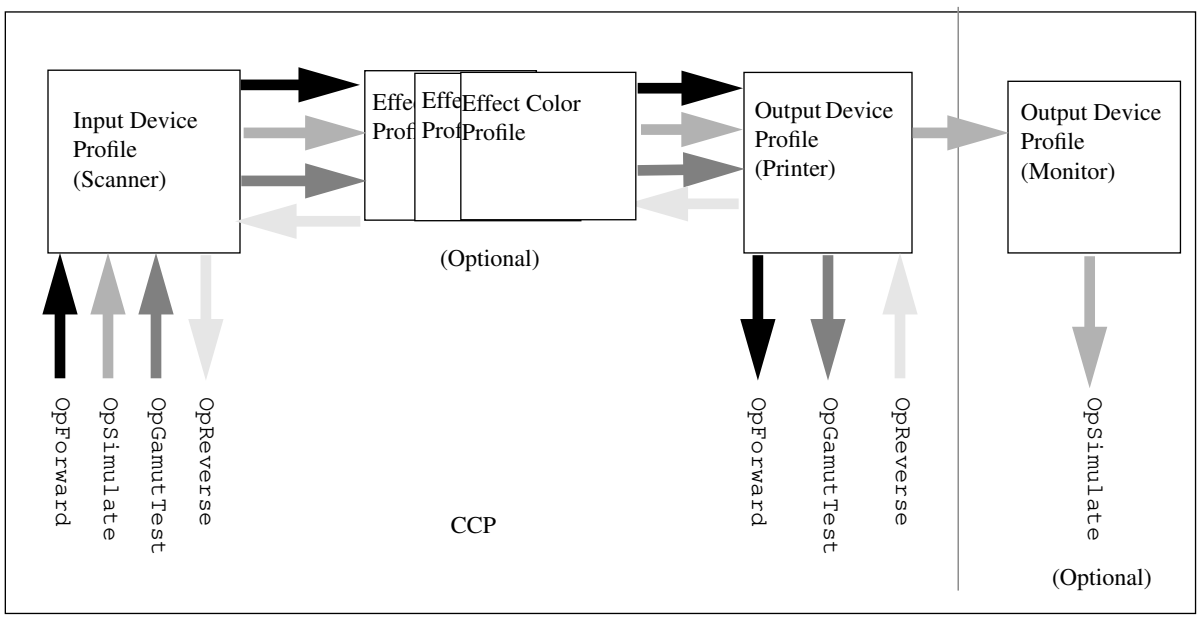

Figure 2-3 Profile Load Hint Operations

- `OpForward`—The forward operation is used to transform color from the scanner form to the monitor form.
- `OpReverse`—The reverse operation is used to transform color from the monitor form to the scanner form. This is useful if your application modifies some colors in monitor space, to keep the greatest number of colors that can be converted back and stored in the scanner's color space.

  A more familiar use of the reverse operation is to transform the color from printer to monitor form to see what the data looks like from the printer.

- `OpSimulate`—The simulate operation is used to simulate the effect of running color data through a CCP, but retaining it in the form of the last device profile. For example, the simulate operation can produce monitor data that simulates the result of printed data.
- `OpGamutTest`—The gamut-test operation is used to determine if each color in the source data is within the gamut of the destination device. Physical devices have a range of colors they can produce. This range of colors is known as the gamut of the device.

`KcsEvaluate()` can take a long time to execute, especially if the input image or graphic contains millions of pixels. Therefore, you can provide a callback function using `KcsSetCallback()`, which `KcsEvaluate()` calls when necessary. The callback function can, for example, provide feedback to request that processing be cancelled. If the callback returns a non-`KCS_SUCCESS` status, the processing stops.

Associating Profiles with Devices

The `KcsSaveProfile()` function, when supplied a `KcsProfileDesc` structure, associates that color profile with the supplied structure. Typically, a configuration or calibration program calls `KcsSaveProfile()`. The profile associated with the `KcsDescription` structure represents the last calibrated condition of the device. For more information about the `KcsSaveProfile()` function, see page 83.

Many events can change the condition of a device. For example, as room lighting changes, so does the viewer's perception of a monitor's colors. As another example, consider a color printer; when different kinds of paper are used in the printer, the printer's color condition changes. When conditions change, a user may associate a different profile with the device.

Using Color Space Profiles

Another possible use of `KcsConnectProfiles()` is to connect a DCP and a CSP, creating a new CCP. Refer to Figure 2-1 on page 12. If the scanner DCP in this figure is connected to the CSP (instead of the Monitor DCP shown that converts for the CIE XYZ color space), the resulting CCP will convert color data produced by the scanner into CIE XYZ format.

Code Example 2-2 shows the sequence of calls that creates and applies the CCP. Note that this example is very similar to Code Example 2-1 on page 13. The difference is the second call to `KcsLoadProfile()`. In Code Example 2-2, `KcsLoadProfile()` loads the CIE XYZ profile description instead of the monitor description.

Code Example 2-2 Connecting a DCP and CSP

```
/*Load scanner's DCP. */
KcsLoadProfile(&inProfile, &scannerDescription, KcsLoadAllNow);
if(status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    KcsFreeProfile(profileid);
    exit(1);
}

/*Load CSP for CIE XYZ color space. */
KcsLoadProfile(&outProfile,&CIEXYZdescription, KcsLoadAllNow);
if(status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    KcsFreeProfile(profileid);
    exit(1);
}

/*Connect two profiles to form a CCP.*/
profileSequence[0] = inProfile;
profileSequence[1] = outProfile;
KcsConnectProfiles(&completeProfile,2, profileSequence,
       KcsLoadAllNow, &failedProfileIndex);
if(status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    KcsFreeProfile(profileid);
    exit(1);
}

/*Apply the CCP to input color data.*/
KcsEvaluate(completeProfile, KcsOperationForward,
       &inbufLayout, &outbufLayout);
```

2

Advanced Profile Topics

Operation Hints

`KcsEvaluate()` takes an additional argument that describes the operation to be performed on the profile; it is an operation hint. For example, you can tell `KcsEvaluate()` to convert data in the forward direction (`KcsOpForward`), such as from the scanner to the printer. You may also convert data in the reverse direction, such as from the monitor to the scanner. The reverse operation (`KcsOpReverse`), when it is available in a profile, inverts the function performed by `KcsOpForward`. However, it rarely performs an exact inverse because information is lost when color data is transformed. In other words, if you perform a `KcsOpForward` and then a `KcsOpReverse` of a profile on the same buffer, the result is almost equivalent to what you started with before `KcsOpForward`; some quality may be lost.

Only one of these operation hint bits can be set at one time for `KcsEvaluate()`, unlike general load hints, for which any combination can be set at the same time. As part of the `KcsLoadHints` data type, the operation hints signify the required set of operations available to use with the profile. By contrast, `KcsEvaluate()` uses only the single operation that the application wants to perform.

See “Operation Hint Constants” on page 43 for more information on operation hints.

Content Hints

You can also specify hints about the content of the data being processed; for example, photographic image data or computer-generated graphic image data. A CMM can use these hints to do a better job of converting the data; for example, the CMM can use these hints to adjust the gamut-mapping technique.

See “Content Hint Constants” on page 44 for more information on content hints.

Freeing Profiles

After creating a complete profile, you can use it more than once; for example, to convert images page-by-page during printing and to process individual rasters or tiles in a large image. Once your program no longer needs the profile, use `KcsFreeProfile()` to free the profile's resources. The profiles in the profile sequence used to create a CCP can be freed without affecting the CCP.

Managing Profile Memory

The C API expects the application to allocate memory required for the data returned by the KCMS framework. In general, the application allocates a C structure and passes a pointer to that structure into the KCMS framework.

The one exception to this is the profile; the KCMS framework returns and accepts an identifier only. Memory allocated for the identifier must be managed by your application. Call `KcsFreeProfile()` in your application to inform the KCMS framework to release the memory associated with the profile identifier.

Optimizing Profiles

Once a color profile has been loaded, a CMM may be able to optimize it. With `KcsOptimizeProfile()`, you can optimize a profile (an individual profile or a CCP) in two ways:

- First, you can optimize a profile to make it more accurate (by eliminating intermediate round-off errors, for instance), smaller (by merging sequences of look-up tables, for instance), or faster (by precomputing some results). The application specifies whether size, speed, accuracy, or some combination is more important.
- Second, by using load hints to limit a profile's operations, you may also affect its optimization. This is valuable, for instance, if you want to write color data with a DCP that will be used later to read the data. The size of the DCP can be significantly reduced (depending on the CMM in use) by restricting the profile to the forward operation only.

Because optimization can take a long time, the application can provide a callback, similar to the one used with `KcsEvaluate()`.

After optimizing a profile, call `KcsSaveProfile()` to save it for future use. Then use this profile with `KcsLoadProfile()` to avoid the slow performance of `KcsOptimizeProfile()`.

There are some potential implications when saving an optimized profile. The optimization may indirectly affect future operations on the profile. For example, if the profile is optimized for size, portions of the profile needed only for highest accuracy may be discarded, resulting in compromised accuracy.

Note – Another way to *optimize* a profile is to use the operation flags supplied when using `KcsConnectProfiles()`. `KcsConnectProfiles()` allows you to specify which operations and content hints should be supported when you connect a sequence of profiles. Reducing the set of operations by supplying the content hint can make `KcsConnectProfiles()` run faster and make the resulting profile smaller.

Characterizing and Calibrating Profiles

Characterization establishes a norm for a particular device across a range of samples of the device. This form of profile is typically supplied by a Profile vendor. To obtain an optimally accurate DCP for a particular device, calibration is required.

Calibration makes measurements of an individual device and applies them to the base DCP. The updated DCP represents the actual color device the customer is using.

The KCMS API provides two API functions, `KcsCreateProfile()` and `KcsUpdateProfile()`, to create new blank profiles and then update them with characterization data or calibration data.

The first step in building a new profile is to create an empty profile using `KcsCreateProfile()`. Fill the empty profile with `KcsSetAttribute()` to describe the device being characterized; for example, supply monitor chromaticities and white-point values. Measurement data is required for `KcsUpdateProfile()` to complete the creation of the new profile. Once updated, save the profile with `KcsSaveProfile()` to the desired `KcsProfileDesc` location.

Updating profiles is typically a CMM-dependent operation. The use of measurement data at the KCMS framework interface level frees you from details of the profile format and the process by which the CMM turns the measurement data into its methodology for color correction.

The default CMM supports characterization and calibration of monitors and scanners.

2