Functions 4

This chapter describes in detail each C API function you can use in applications. It describes each function's signature, use, arguments, and return values. For several functions, the chapter provides code examples. The functions are defined in the `kcs.h` header file and are presented in alphabetical order.

All constants, definitions, macros, and data types are defined in Chapter 3, "Data Structures," Chapter 5, "KCMS Profile Attributes and ICC Tags" and in the ICC specification. By default, the ICC specification is located online in the `SUNWsdk/kcms/doc` directory.

These API functions support error and warning messages returned by the operating system. See Chapter 6, "Warning and Error Messages," for all error and warning messages returned by these functions.

4

KcsAvailable()

```
KcsStatusId
KcsAvailable(long *response)
```

Purpose The `KcsAvailable()` function determines if the KCMS framework has been installed on the system. This function is provided primarily for cross-platform compatibility.

Arguments `response` is a pointer to a `long` for temporary use in the `KcsAvailable` function.

Returns `KCS_SUCCESS` is always returned in the Solaris environment.

KcsConnectProfiles()

```
KcsStatusId
KcsConnectProfiles(KcsProfileId *resultProfileId,
        unsigned long profileCount,
        KcsProfileId *profileSequence,
        KcsOperationType operationLoadSet,
        unsigned long *failedProfileIndex)
```

Purpose

Use `KcsConnectProfiles()` to combine several existing profiles into a new complete profile, or to restrict the functionality of a single existing profile to make it more efficient.

If `KcsConnectProfiles()` returns successfully, it generates a new profile from the sequence of existing profiles. The reference to this new profile is stored in the `resultProfileId` argument. With this reference, you can free the resources of the existing profiles in `profileSequence` if they are no longer required. Use `KcsFreeProfile()` to release the resources.

Note – If you have minimized a profile's load operation or state with `operationLoadSet` or with `KcsOptimizeProfile()` (page 81), only that load operation or state is saved with `KcsSaveProfile()`. Therefore, operations not included in the profile are not available the next time the profile is loaded.

Arguments

`resultProfileId`

The profile returned if the function executes successfully.

`profileCount`

The number of profiles to be connected.

`profileSequence`

An array of the identifiers of the profiles to be connected.

`operationLoadSet`

One or more flags symbolizing the kind of information in the resultant profile. It also describes what, how, when, and where to load and unload the resulting `resultProfileId`. See "KcsLoadHints" on page 40 for more information.

`failedProfileIndex`

`KcsConnectProfiles()` returns an integer in `failedProfileIndex`. This value has meaning only when `KcsConnectProfiles()` returns a value other than `KCS_SUCCESS`. If the function fails, this index helps you identify which input profile caused the failure. If the index = `0`, the first profile in `profileSequence` failed; if it = `1`, the second profile in `profileSequence` failed, and so on. A common problem when making the resultant profile is that the profiles specified in `profileSequence` could not be connected. In this case, the index returns an integer symbolizing the latter profile in a failed connection pair. For example, if the first profile and second profile in the sequence were mismatched, the index contains 1 (for the second profile).

Returns

`KCS_SUCCESS`
`KCS_PROF_ID_BAD`
`KCS_MEM_ALLOC_ERROR`
`KCS_CONNECT_PRECISION_UNACCEPTABLE`
`KCS_MISMATCHED_COLORSPACES`
`KCS_CONNECT_OPT_FORCED_DATA_LOSS`

Example

Code Example 4-1 `KcsConnectProfiles()`

```
KcsProfileDesc scannerDesc, monitorDesc, completeDesc;
KcsProfileId   scannerProfile, monitorProfile;
KcsProfileId   profileSequence[2], completeProfile;
KcsStatusId    status;
KcsErrDesc     errDesc;
u_long         failedProfileNum;
KcsOperationType=(KcsOpForward+KcsContImage);
/*file names input a program arguments */

scannerDesc.type = KcsSolarisProfile;
```

Code Example 4-1 `KcsConnectProfiles()` *(Continued)*

```
scannerDesc.desc.solarisFile.fileName = argv[1];
scannerDesc.desc.solarisFile.hostName = NULL;
scannerDesc.desc.solarisFile.oflag = O_RDONLY;
scannerDesc.desc.solarisFile.mode = 0;

monitorDesc.type = KcsSolarisProfile;
monitorDesc.desc.solarisFile.fileName = argv[2];
monitorDesc.desc.solarisFile.hostName = NULL;
monitorDesc.desc.solarisFile.oflag = O_RDONLY;
monitorDesc.desc.solarisFile.mode = 0;

status = KcsLoadProfile(&scannerProfile, &scannerDesc, KcsLoadAllNow);

if(status != KCS_SUCCESS) {
    KcsGetLastError(&errDesc);
    printf("Scanner LoadProfile error: %s\n", errDesc.desc);
    exit(1);
}

status = KcsLoadProfile(&monitorProfile, &monitorDesc, KcsLoadAllNow);

if(status != KCS_SUCCESS) {
    KcsGetLastError(&errDesc);
    printf("Monitor LoadProfile error: %s\n", errDesc.desc);
    exit(1);
}

/* See if we can combine them */
profileSequence[0] = scannerProfile;
profileSequence[1] = monitorProfile;

status = KcsConnectProfiles(&completeProfile, 2, profileSequence, op,
        &failedProfileNum);

if(status != KCS_SUCCESS) {
    KcsGetLastError(&errDesc);
    printf("ConnectProfile error: %s\n", errDesc.desc);
    fprintf(stderr, "Failed in profile number %d\n", failedProfileNum);
    exit(1);
}
```

4

KcsCreateProfile()

```
KcsStatusId
KcsCreateProfile(KcsProfileId *resultProfileId,
        KcsCreationDesc *desc)
```

Purpose Use `KcsCreateProfile()` to create an empty profile. The profile will contain neither attributes nor CMM-specific data.

Note – Currently, you cannot call `KcsGetAttribute()` for a list of the installed and available CMMs. The workaround is to load all available profiles and do a `KcsGetAttribute()` for each individual CMM type.

Arguments `resultProfileId`

The reference to the resultant profile, returned if the function executes successfully.

`desc`

This is a pointer to a `KcsCreationDesc` (see page 36) structure that describes the static store used to save the profile and an extendable union of profile information used to create the profile. The `id` member of the union describes which CMM and version to use, and the profile format and version to use.

If `desc` is `NULL` the default CMM and profile format are used.

Returns `KCS_SUCCESS`
`KCS_MEM_ALLOC_ERROR`

Example

Code Example 4-2 `KcsCreateProfile()`

```
KcsProfileDesc     desc;
KcsCreationDesc    c_desc;
KcsProfileId       profileid;
KcsStatusId        status;
KcsErrDesc         errDesc;
/* The filename is a command line argument */
/* Create a new profile with the default CMM */

desc.type = KcsSolarisProfile;
desc.desc.solarisFile.fileName = argv[1];
desc.desc.solarisFile.hostName = NULL;
desc.desc.solarisFile.oflag = O_RDWR|O_CREAT|O_TRUNC;
desc.desc.solarisFile.mode = 0666;
c_desc.profileDesc = &desc;
c_desc.desc.id.cmmId = 0;
c_desc.desc.id.cmmVersionId = 0;
c_desc.desc.id.profileId = 0;
c_desc.desc.id.profileVersionId = 0;
status = KcsCreateProfile(&profileid, &c_desc);
if(status != KCS_SUCCESS) {
    KcsGetLastError(&errDesc);
    printf("CreateProfile error: %s\n", errDesc.desc);
}
```

Note – Other required fields in the profile must be set with `KcsSetAttribute()`.

4

KcsEvaluate()

```
KcsStatusId
KcsEvaluate(
        KcsProfileId profile,
        KcsOperationType operation,
        KcsPixelLayout *srcData,
        KcsPixelLayout *destData)
```

Purpose Use `KcsEvaluate()` to apply a color profile to input color data to produce color-corrected output data.

See "KcsPixelLayout" on page 49 for more information about using pixel layouts in this context.

Arguments `profile`

The ID of the profile to be applied to the input data. If the operation specified when the profile was created in `KcsConnectProfiles()` does not match the operation specified in `KcsEvaluate()`, the `statusId` `KCS_EVAL_ONLY_ONE_OP_ALLOWED` is returned. If, for example, you wanted to evaluate forward (specified `KcsOpForward` in `KcsEvaluate()`) with a profile you created with `KcsConnectProfiles()` to simulate (used `KcsOpSimulate` in `KcsConnectProfiles()`), the above `statusId` would be returned.

`operation`

The kind of data to be operated on, and the kind of profile evaluation to be performed on, the data. (See "Operation Hint Constants" on page 43 and "Content Hint Constants" on page 44 more information.) Note that only one bit can be set for `KcsEvaluate()`.

`srcData`

The description of the source color data to be transformed by the profile.

`destData`

The description of the area (the destination) to which the transformed data is written.

Returns

`KCS_SUCCESS`
`KCS_OPERATION_CANCELLED`
`KCS_PROF_ID_BAD`
`KCS_MEM_ALLOC_ERROR`
`KCS_EVAL_ONLY_ONE_OP_ALLOWED`
`KCS_EVAL_TOO_MANY_CHANNELS`
`KCS_EVAL_BUFFER_OVERFLOW`
`KCS_LAYOUT_INVALID`
`KCS_LAYOUT_UNSUPPORTED`
`KCS_LAYOUT_MISMATCH`

Example

Code Example 4-3 `KcsEvaluate()`

```
int                 op;
KcsPixelLayout      pixelLayoutIn, pixelLayoutOut;
KcsProfileId        scannerProfile, monitorProfile;
KcsProfileId        profileSequence[2], completeProfile;

/* Load and connect profiles. */
/* Load input and output pixel layout structures with appropriate data. */

status = KcsEvaluate(completeProfile, op, &pixelLayoutIn,
           &pixelLayoutOut);
```

KcsFreeProfile()

```
KcsStatusId
KcsFreeProfile(
        KcsProfileId    profile)
```

Purpose Use `KcsFreeProfile()` to release all resources a loaded profile is using. A loaded profile uses memory and additional types of resources.

The KCMS framework does not automatically save profile changes when your application terminates. To save profile changes, you must call `KcsSaveProfile()`.

Note – If the application passes a `KcsFileProfile` type of `KcsProfileDesc` as an argument, `KcsFreeProfile()` does not close the `KcsFileId` contained in the file entry of the `KcsProfileDesc` union.

Arguments `profile`

The identifier of a loaded profile.

Returns `KCS_SUCCESS`
`KCS_PROF_ID_BAD`

Example

Code Example 4-4 `KcsFreeProfile()`

```
KcsProfileId   profile;

/* Complete all processing. */

KcsFreeProfile(profile);
```

KcsGetAttribute()

```
KcsStatusId
KcsGetAttribute(KcsProfileId profile, KcsAttributeName name,
        KcsAttributeValue *value)
```

Purpose Use `KcsGetAttribute()` to find the value of a particular attribute of the given profile. (See Chapter 5, "KCMS Profile Attributes and ICC Tags" for more information on attributes.)

Arguments

`profile`

The identifier of the loaded profile.

`name`

ICC tag name.

`value`

A pointer to the structure to hold the value of the profile's attribute. You need to set the `countSupplied` field in the `value` argument. If you do not set it, you may see the warning `KCS_ATTR_LARGE_CT_SUPPLIED` or the error `KCS_ATTR_CT_ZERO_OR_NEG` returned.

Returns

`KCS_SUCCESS`
`KCS_PROF_ID_BAD`
`KCS_ATTR_NAME_OUT_OF_RANGE`
`KCS_ATTR_CT_ZERO_OR_NEG`
`KCS_ATTR_LARGE_CT_SUPPLIED` (warning)

Example

Code Example 4-5 `KcsGetAttribute()`

```
#include "./kcms_utils.h"

KcsProfileId       profileid;
KcsAttributeValue  *attrValue;
int                size;
void print_header(icHeader *hdr);

size = sizeof(KcsAttributeBase) + sizeof(icHeader);
```

Code Example 4-5 `KcsGetAttribute()` *(Continued)*

```
attrValue = (KcsAttributeValue *)malloc(size);

/* Get the header */
attrValue->base.type = icSigHeaderType;
attrValue->base.sizeOfType = sizeof(icHeader);
attrValue->base.countSupplied = 1;
KcsGetAttribute(profileid, icSigHeaderTag, attrValue);
...
print_header(&attrValue->val.icHeader);
...

void
print_header(icHeader *hdr)
{
    char    charstring[5];

    printf("Size in bytes = %d\n", hdr->size);
    printf("CMM Id = 0x%x\n", hdr->cmmId);
    printf("Major version number = 0x%x\n", hdr->version>>24);
    printf("Minor version number = 0x%x\n", (hdr->version&0x00FF0000)>>16);

    switch(hdr->deviceClass) {
    case icSigInputClass :
        printf("deviceClass = input\n");
        break;
    case icSigDisplayClass :
        printf("deviceClass = display\n");
        break;
    case icSigOutputClass :
        printf("deviceClass = output\n");
        break;
    case icSigLinkClass :
        printf("deviceClass = link\n");
        break;
    case icSigAbstractClass :
        printf("deviceClass = abstract\n");
        break;
    case icSigColorSpaceClass :
        printf("deviceClass = colorspace\n");
        break;
    default :
        printf("Unknown\n");
        break;
```

Code Example 4-5 `KcsGetAttribute()` *(Continued)*

```
    }

    memset(charstring, 0 ,5);
    memcpy(charstring, &hdr->colorSpace, 4);
    printf("colorspace = %s\n", charstring);

    memset(charstring, 0 ,5);
    memcpy(charstring, &hdr->pcs, 4);
    printf("profile connection space = %s\n", charstring);

    printf("date = %d/%d/%d,  ", hdr->date.day,hdr->date.month,
           hdr->date.year);
    printf("time = %d:%d:%d\n", hdr->date.hours,hdr->date.minutes,
           hdr->date.seconds);

    memset(charstring, 0 ,5);
    memcpy(charstring, &hdr->magic, 4);
    printf("magic number = %s\n", charstring);

    switch(hdr->platform) {
    case icSigMacintosh :
        printf("platform = Macintosh\n");
        break;
    case icSigMicrosoft :
        printf("platform = Microsoft\n");
        break;
    case icSigSolaris :
        printf("platform = Solaris\n");
        break;
    case icSigSGI :
        printf("platform = SGI\n");
        break;
    case icSigTaligent :
        printf("platform = Taligent\n");
        break;
    default :
        printf("Unknown\n");
        break;
    }

    if(hdr->flags && icEmbeddedProfileTrue)
        printf("Embedded profile.\n");
    else
        printf("Non-embedded profile\n");
```

Code Example 4-5 `KcsGetAttribute()` *(Continued)*

```
    if(hdr->flags && icUseWithEmbeddedDataOnly)
        printf("If this profile is embedded, it is not allowed to strip
                it out and use it independently.\n");
    else
        printf("OK to strip embedded profile out and use
                independently\n");

    memset(charstring, 0 ,5);
    memcpy(charstring, &hdr->manufacturer, 4);
    printf("manufacturer = %s\n", charstring);

    printf("model number = %d\n", hdr->model);

    printf("device attributes = %d%d\n", hdr->attributes[0],
            hdr->attributes[1]);

    switch (hdr->renderingIntent) {
    case 0 :
        printf("rendering intent = Perceptual\n");
        break;
    case 1 :
        printf("rendering intent = Relative Colorimetric\n");
        break;
    case 2 :
        printf("rendering intent = Saturation\n");
        break;
    case 3 :
        printf("rendering intent = Absolute Colorimetric\n");
        break;
    default :
        printf("Unknown\n");
        break;
    }

    printf("Illuminat X=%f Y=%f X=%f\n",
        icfixed2double(hdr->illuminant.X, icSigS15Fixed16ArrayType),
        icfixed2double(hdr->illuminant.Y, icSigS15Fixed16ArrayType),
        icfixed2double(hdr->illuminant.Z, icSigS15Fixed16ArrayType));
}
```

KcsGetLastError()

```
KcsStatusId
KcsGetLastError (KcsErrDesc *errDesc)
```

Purpose Use `KcsGetLastError()` to find information about the most recent error.

Arguments `errDesc`

A pointer to the structure holding information about the last error.

If an operating-system-defined error occurs, the `sysErrNo` field is set.

The `desc` field contains the description of the particular `statId`. This is either a string in Table 6-1 or Table 6-2 on page 136, or the literal string “Internal Color Processor Error” or “No description for this status id number”. See Chapter 6, “Warning and Error Messages” for information on using `KcsGetLastError()` to localize `KcsStatusId`.

Returns `KCS_SUCCESS`

Example

Code Example 4-6 `KcsGetLastError()`

```
KcsErrDesc errDesc;

status = KcsLoadProfile(&profile, &profileDesc,
        KcsLoadAttributesNow);
if (status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    fprintf(stderr,"%s KcsLoadProfile failed error = %s\n",
            errDesc.desc);
    exit(1);
}
```

4

KcsLoadProfile()

```
KcsStatusId
KcsLoadProfile(KcsProfileId *profile,
       KcsProfileDesc *desc, KcsLoadHints loadHints)
```

Purpose

Use `KcsLoadProfile()` to load a profile and all of its resources into the system.

The function uses `desc` to determine where to get the data to generate the profile's resources in the system. (See page 54 for an in-depth description of `KcsProfileDesc`.) It uses `profile` to return a reference to the loaded profile; this reference is needed by other API functions.

You can determine the length of the data read from the file by calling `KcsGetAttribute()` and supplying the `icHeader` attribute. The value of `size` in `icHeader` is the size of the profile. (For the format of the icHeader structure, see "icHeader" on page 132.)

With the `loadHints` argument, `KcsLoadProfile()` allows the application to suggest how the KCMS framework manages the memory and other resources associated with a loaded profile. Although this is a flexible mechanism, these caveats apply:

- The load hints are merely hints, which means the KCMS framework can ignore them. However, because the functionalities of various CMMs loaded by the KCMS framework cannot always be determined, your application should supply the load hints anyway. Furthermore, even if a CMM loaded by the KCMS framework does not support a particular load hint in its current release, it may support it in future releases.
- If the application supplies a hint that indicates that the profile is to be loaded at a time other than now, it must keep the described mechanism open to allow for data access at a future and somewhat arbitrary time. For example, if the application specifies `KcsLoadWhenNecessary` and the `desc` argument describes a file, and the application uses a `KcsFileId`, it

cannot close the file until it first frees the profile. This allows the KCMS framework to read any necessary data to load the profile at any time.

After you are finished with the profile, call `KcsFreeProfile()` to release the resources allocated by this profile.

Note – If you use the `KcsFileId` entry in the file part of the `KcsProfileDesc` union, `KcsFileId` marks the *current position* within an open file. After a call to `KcsLoadProfile()`, the current position is undefined. The application must reset the pointer before doing any other I/O.

Arguments

`profile`

The identifier of the profile returned after the profile is loaded into memory. This value serves as an argument to all other functions, such as `KcsEvaluate()`.

`desc`

The location of the profile's static storage, needed to obtain the data required to generate the profile's resources. It is specified as a union of independent static storage mechanisms. The `KcsProfileDesc` structure has a field that identifies which storage mechanism to use.

`loadHints`

The set of bits describing what, how, when and where to load and unload `profile`. See page 40 for more information on the `KcsLoadHints` data type.

Returns

KCS_SUCCESS
KCS_MEM_ALLOC_ERROR
KCS_IO_READ_ERR
KCS_IO_SEEK_ERR
KCS_SOLARIS_FILE_NOT_OPENED
KCS_SOLARIS_FILE_RO
KCS_SOLARIS_FILE_LOCKED
KCS_SOLARIS_FILE_NAME_NULL
KCS_X11_DATA_NULL
KCS_X11_PROFILE_NOT_LOADED
KCS_X11_PROFILE_RO

Example

Code Example 4-7 KcsLoadProfile()

```
#pragma ident "@(#)kcstest.c1.8  11/28/94"
/*
 * Copyright (c) 1994, by Sun Microsystems, Inc.

    KcsFileIdscannerFd, monitorFd, completeFd;
    KcsProfileDescscannerDesc, monitorDesc, completeDesc;
    KcsProfileId scannerProfile, monitorProfile;
    KcsProfileIdprofileSequence[2], completeProfile;
    KcsStatusIdstatus;
    KcsAttributeValueattrValue;
    KcsAttributeNamei;
    KcsOperationTypeop = (KcsOpForward+KcsContImage);
    u_longfailedProfileNum;
    extern voidkcs_timer(int);

    if (argc > 4) {
    fprintf(stderr,"Usage : kcstest profile_1 profile_2 [save_profile]\n");
    exit(1);
    }

#ifdef FILE_DESC
    /* Open up the files from disk */
    scannerDesc.type = KcsFileProfile;
    scannerFd = open(argv[1], O_RDONLY);
    if (scannerFd == -1) {
    perror("Failed to open scanner profile");
    exit(1);
    }
    scannerDesc.desc.file.openFileId = scannerFd;
```

Code Example 4-7 `KcsLoadProfile()`

```
    scannerDesc.desc.file.offset = 0;

    monitorDesc.type = KcsFileProfile;
    monitorFd = open(argv[2], O_RDONLY);
    if (monitorFd == -1) {
    perror("Failed to open monitor profile");
    exit(1);
    }
    monitorDesc.desc.file.openFileId = monitorFd;
    monitorDesc.desc.file.offset = 0;
#endif

#ifdef FILE_NAME
    scannerDesc.type = KcsSolarisProfile;
    scannerDesc.desc.solarisFile.fileName = argv[1];
    scannerDesc.desc.solarisFile.hostName = NULL;
    scannerDesc.desc.solarisFile.oflag = O_RDONLY;
    scannerDesc.desc.solarisFile.mode = 0;

    monitorDesc.type = KcsSolarisProfile;
    monitorDesc.desc.solarisFile.fileName = argv[2];
    monitorDesc.desc.solarisFile.hostName = NULL;
    monitorDesc.desc.solarisFile.oflag = O_RDONLY;
    monitorDesc.desc.solarisFile.mode = 0;
#endif

    /* Load the profiles */
    printf("Load scanner profile\n");
    kcs_timer(START);
    status = KcsLoadProfile(&scannerProfile, &scannerDesc, KcsLoadAllNow);
    kcs_timer(STOP);
    if (status != KCS_SUCCESS) {
    fprintf(stderr,"Scanner KcsLoadProfile failed error = 0x%x\n", status);
#ifdef FILE_DESC
    close(scannerFd);
    close(monitorFd);
#endif
    exit(1);
    }

    printf("Load monitor profile\n");
    kcs_timer(START);
    status = KcsLoadProfile(&monitorProfile, &monitorDesc, KcsLoadAllNow);
    kcs_timer(STOP);
```

Code Example 4-7 `KcsLoadProfile()`

```
    if (status != KCS_SUCCESS) {
    fprintf(stderr,"MonitoKcsLoadProfile failed error = 0x%x\n", status);
#ifdef FILE_DESC
    close(scannerFd);
    close(monitorFd);
#endif
    exit(1);
    }
```

KcsModifyLoadHints()

```
KcsStatusId
KcsModifyLoadHints(KcsProfileId profile,
        KcsLoadHints newHints)
```

Purpose `KcsModifyLoadHints()` applies a new set of load hints to a profile already loaded. If, for example, you no longer need to simulate a profile and available memory is limited, you can use this function to unload the simulation portion of the profile immediately, making more memory available for the application.

Note – Remember that the load hints are just that—hints to the KCMS framework. Although the KCMS framework tries to accomplish what is specified, and typically does, it cannot guarantee everything exactly as hinted.

Arguments `profile`

The identifier of a loaded profile.

`newHints`

The set of bits describing what, how, when, and where to load and unload `profile`. See "KcsLoadHints" on page 40 for more information.

Returns `KCS_SUCCESS`
`KCS_PROF_ID_BAD`
`KCS_MEM_ALLOC_ERROR`

Example

Code Example 4-8 `KcsModifyLoadHints()`

```
KcsProfileId   profileid;
KcsErrDesc     errDesc;
KcsProfileDesc profileDesc;
KcsProfileId   profile;
KcsStatusId    status;
KcsLoadHints   newhints;

/* profile name is a command line argument */
```

Code Example 4-8 `KcsModifyLoadHints()` *(Continued)*

```
profileDesc.type = KcsSolarisProfile;
profileDesc.desc.solarisFile.fileName = argv[1];
profileDesc.desc.solarisFile.hostName = NULL;
profileDesc.desc.solarisFile.mode = 0;
profileDesc.desc.solarisFile.oflag = NULL;

status = KcsLoadProfile(&profile, &profileDesc, KcsLoadAttributesNow);
if (status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    fprintf(stderr,"%s KcsLoadProfile failed error = %s\n",
        argv[optind], errDesc.desc);
    exit(1);
}

/* suppose it was determined that this is the profile we want to *
 * use for evaluating data. We want to load it all in now. */

newhints = KcsLoadAllNow;
status = KcsModifyLoadHints(profile, newhints);
if (status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    fprintf(stderr," ModifyHints failed error = %s\n", errDesc.desc);
    exit(1);
}
```

KcsOptimizeProfile()

```
KcsStatusId
KcsOptimizeProfile(KcsProfileId profile,
        KcsOptimizationType optimizationType,
        KcsLoadHints operationLoadSet)
```

Purpose Use `KcsOptimizeProfile()` to optimize the profile by:

- Reducing the profile's size
- Increasing the profile's speed
- Increasing the profile's accuracy

Optimization is CMM dependent. The CMM always interprets the load hints in terms of the particular situation.

Note – If you have minimized a profile's load operation or state with `operationLoadSet` or with `KcsOptimizeProfile()`, only that load operation or state is saved with `KcsSaveProfile()`. Therefore, operations not included in the profile are not available the next time the profile is loaded.

Arguments `profile`

The identifier of the profile.

`optimizationType`

The kinds of optimization (size, speed, and accuracy) you want to perform on the profile. (See "KcsOptimizationType" on page 48 for more information.) When a combination of values is specified, it is up to the CMM to determine which value is more important.

`operationLoadSet`

One or more flags symbolizing the kind of information wanted in `profile`. It also describes what, how, when, and where to load and unload `profile`. See "KcsLoadHints" on page 40 for more information.

Returns

KCS_SUCCESS
KCS_OPERATION_CANCELLED
KCS_MEM_ALLOC_ERROR
KCS_PROF_ID_BAD

Example

Code Example 4-9 KcsOptimizeProfile()

```
KcsProfileId   monitorProfile, scannerProfile, completeProfile;
KcsStatusId    status;
KcsErrDesc     errDesc;

/* The monitor profile and scanner profile have been loaded and connected *
 * to become a complete profile, now optimize. */

status = KcsOptimizeProfile(completeProfile, KcsOptSpeed, KcsLoadAllNow);
if (status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    fprintf(stderr,"KcsOptimizeProfile failed error = %s\n", errDesc.desc);
    KcsFreeProfile(monitorProfile);
    KcsFreeProfile(scannerProfile);
    return(-1);
}
```

KcsSaveProfile()

```
KcsStatusId
KcsSaveProfile (KcsProfileId profile, KcsProfileDesc *desc)
```

Purpose

Use `KcsSaveProfile()` to save a loaded profile, and any changes to its attributes or profile data, to the mechanism described by `desc`.

If supported by the mechanism, a profile's state can be saved at an offset. For example, if the mechanism indicates a file, the following two situations are applicable:

- Create a file containing only one profile. In this case most typically the offset would be 0.
- Create a file containing one profile plus some application data (like a TIFF file). You must ensure that the profile fits into the file format and does not overwrite data nor is itself overwritten. You can determine the length of the data read from the file by calling `KcsGetAttribute()` and supplying the `icHeader` attribute. The value of `size` in `icHeader` is the size of the profile.

`KcsSaveProfile()` writes information, but does not free the profile. Even after saving the profile, the application can continue to use it. In fact, the application must call `KcsFreeProfile()` to free all resources associated with the profile.

Arguments

`profile`

The identifier of the loaded profile. Typically, your application obtains this value when it calls `KcsLoadProfile()` or `KcsConnectProfiles()`.

`desc`

The location of the profile's static storage mechanism, needed to obtain the data required to generate the profile's resources. It is specified as a union of independent static storage mechanisms. This argument has a field that identifies which storage

mechanism to use. If this field is NULL, the profile is saved through the same mechanism from which it was loaded. (See "KcsProfileId" on page 56 for more information.)

Returns

KCS_SUCCESS
KCS_IO_WRITE_ERR
KCS_IO_READ_ERR
KCS_IO_SEEK_ERR
KCS_SOLARIS_FILE_NOT_OPENED
KCS_SOLARIS_FILE_RO
KCS_SOLARIS_FILE_LOCKED
KCS_SOLARIS_FILE_NAME_NULL
KCS_X11_DATA_NULL
KCS_X11_PROFILE_NOT_LOADED
KCS_X11_PROFILE_RO

Example

Code Example 4-10 KcsSaveProfile()

```
KcsProfileDesc desc;
KcsProfileId   profileid;
KcsStatusId    status;
KcsErrDesc     errDesc;

/*see example kcms_update.c for a full example code */

desc.type = KcsSolarisProfile;
desc.desc.solarisFile.fileName = argv[1];
desc.desc.solarisFile.hostName = NULL;
desc.desc.solarisFile.mode = 0;
desc.desc.solarisFile.oflag = O_RDWR
status = KcsSaveProfile(profileid, &desc);
if(status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    fprintf(stderr,"KcsSaveProfile failed error = %s\n", errDesc.desc);
}
KcsFreeProfile(profileid);
```

Note – If you are saving a *new* profile, use the following assignments instead of the assignments in Code Example 4-10:

```
desc.desc.solarisFile.mode = 0666;
desc.desc.solarisFile.oflag = O_RDWR | O_CREAT | O_TRUNC;
```

4

KcsSetAttribute()

```
KcsStatusId
KcsSetAttribute(KcsProfileId profile,
        KcsAttributeName name,
        KcsAttributeValue *value)
```

Purpose Use `KcsSetAttribute()` to create, to modify, or to delete a specific attribute in a profile. See Chapter 5, "KCMS Profile Attributes and ICC Tags" for details on attributes.

Arguments `profile`

The identifier to the profile.

`name`

The attribute to be created, modified, or deleted. If this attribute is already used in the profile, this function overwrites its value. If this attribute does not already exist, the function creates it. See the attribute tables in Chapter 5, "KCMS Profile Attributes and ICC Tags" for more information.

`value`

A pointer to the value for the attribute. If the attribute already exists, then `value` becomes the attribute's new value. If the attribute does not already exist, this function creates it and sets its original value to `value`. To delete an existing attribute, set `value` to `NULL`.

Note – For this function to execute correctly, you must check what needs to be set in the `KcsAttributeBase` structure (part of the `KcsAttributeValue` structure). A valid type and number of tokens found in the attribute must be set.

Returns
`KCS_SUCCESS`
`KCS_MEM_ALLOC_ERROR`
`KCS_PROF_ID_BAD`
`KCS_ATTR_NAME_OUT_OF_RANGE`

KCS_ATTR_TYPE_UNKNOWN
KCS_ATTR_NEG_CT_SUPPLIED
KCS_ATTR_LARGE_CT_SUPPLIED

Example

Code Example 4-11 KcsSetAttribute()

```
#include "kcms_utils.h"
#define SAMPLE_WORDS "A profile created using kcms_create"

KcsProfileId      profileid;
KcsStatusId       status;
KcsAttributeValue attrValue;
KcsAttributeValue *attrValue2;
KcsAttributeValue *attrValuePtr;
KcsErrDesc        errDesc;
int               sizemeas, size, nvalues, i, j;
time_t            clocktime;
struct tm         *datetime;
size_t            rc;
char              *description;
char              attr[256];
double            test_double[3];

/* Fill out the measurement structures - The illuminant must be D50 */
test_double[0] = 0.9642;
test_double[1] = 1.0;
test_double[2] = 0.8249;

/*  open or create a profile, then set some attributes */
if ((description = (char *)malloc(strlen(SAMPLE_WORDS) + 1)) == NULL) {
    perror("malloc failed : ");
    KcsFreeProfile(profileid);
    exit(1);
}
memset(description, 0, strlen(SAMPLE_WORDS) + 1);
strcpy(description, SAMPLE_WORDS);
/* the function used below can be found in kcms_utils.c in appendix */
if ((attrValue2 = string2icTextAttrValue(description)) == NULL) {
    fprintf(stderr, "conversion to AttrValue failed \n");
    KcsFreeProfile(profileid);
    exit(1);
}
if (KcsSetAttribute(profileid, icSigProfileDescriptionTag, attrValue2)
    != KCS_SUCCESS) {
```

Code Example 4-11 `KcsSetAttribute()` *(Continued)*

```
    KcsGetLastError(&errDesc);
    printf("Set Attribute error: %s\n", errDesc.desc);
    exit(1);
}
free(attrValue2);
free(description);
size = sizeof(KcsAttributeBase) + sizeof(icHeader);
attrValuePtr = (KcsAttributeValue *)malloc(size);

/* Build the header */
attrValuePtr->base.type = icSigHeaderType;
attrValuePtr->base.sizeOfType = sizeof(icHeader);
attrValuePtr->base.countSupplied = 1;
KcsGetAttribute(profileid, icSigHeaderTag, attrValuePtr);
attrValuePtr->val.icHeader.size = 0;

/* The following three values do not have to be set if you do a
 * GetAttribute on the header, since the Create should set them for you.
 * If you do not do a GetAttribute of the header, you must set these:
 *   attrValuePtr->val.icHeader.cmmId = 0x4b434d53;
 *   attrValuePtr->val.icHeader.version =icVersionNumber;
 *   attrValuePtr->val.icHeader.magic = icMagicNumber;
*/
attrValuePtr->val.icHeader.deviceClass = icSigDisplayClass;
attrValuePtr->val.icHeader.colorSpace = icSigRgbData;
attrValuePtr->val.icHeader.pcs = icSigXYZData;

/* Get the time from the system */
clocktime = time(NULL);
datetime = localtime(&clocktime);

attrValuePtr->val.icHeader.date.seconds =
        (icUInt16Number)datetime->tm_sec;
attrValuePtr->val.icHeader.date.minutes =
        (icUInt16Number)datetime->tm_min;
attrValuePtr->val.icHeader.date.hours =
        (icUInt16Number)datetime->tm_hour;
attrValuePtr->val.icHeader.date.day =
        (icUInt16Number)datetime->tm_mday;
attrValuePtr->val.icHeader.date.month =
        (icUInt16Number)datetime->tm_mon + 1;
attrValuePtr->val.icHeader.date.year =
        (icUInt16Number)datetime->tm_year;
attrValuePtr->val.icHeader.platform = icSigSolaris;
```

Code Example 4-11 `KcsSetAttribute()` *(Continued)*

```
attrValuePtr->val.icHeader.flags =
        icEmbeddedProfileFalse || icUseAnywhere;
strcpy(description,"SUNW ");
memcpy(&attrValuePtr->val.icHeader.manufacturer, description, 4);
attrValuePtr->val.icHeader.model = 0;
attrValuePtr->val.icHeader.attributes[0] = 0;
attrValuePtr->val.icHeader.attributes[1] = 0;
attrValuePtr->val.icHeader.renderingIntent = icPerceptual;
attrValuePtr->val.icHeader.illuminant.X =
        double2icfixed(test_double[0], icSigS15Fixed16ArrayType);
attrValuePtr->val.icHeader.illuminant.Y =
        double2icfixed(test_double[1], icSigS15Fixed16ArrayType);
attrValuePtr->val.icHeader.illuminant.Z =
        double2icfixed(test_double[2], icSigS15Fixed16ArrayType);
rc = KcsSetAttribute(profileid, icSigHeaderTag, attrValuePtr);
if(rc != KCS_SUCCESS) {
    rc =KcsGetLastError(&errDesc);
    fprintf(stderr, "unable to set header: %s\n", errDesc.desc);
    KcsFreeProfile(profileid);
    return(-1);
}
free(attrValuePtr);

/* set white point and colorants with dummy values to show it works*/
attrValue.base.countSupplied = 1;
attrValue.base.type = icSigXYZType;
attrValue.base.sizeOfType = sizeof(icXYZNumber);
attrValue.val.icXYZ.data[0].X = double2icfixed(test_double[0],
        icSigS15Fixed16ArrayType);
attrValue.val.icXYZ.data[0].Y = double2icfixed(test_double[1],
        icSigS15Fixed16ArrayType);
attrValue.val.icXYZ.data[0].Z = double2icfixed(test_double[2],
        icSigS15Fixed16ArrayType);
rc = KcsSetAttribute(profileid, icSigMediaWhitePointTag, &attrValue);
if(rc != KCS_SUCCESS) {
    KcsGetLastError(&errDesc);
    fprintf(stderr, "unable to set whitepoint: %s\n", errDesc.desc);
    KcsFreeProfile(profileid);
    return(-1);
}
test_double[0] = 0.572586;
test_double[1] = 0.337198;
test_double[2] = 0.026291;
```

Code Example 4-11 `KcsSetAttribute()` *(Continued)*

```
attrValue.val.icXYZ.data[0].X = double2icfixed(test_double[0],
        icSigS15Fixed16ArrayType);
attrValue.val.icXYZ.data[0].Y =double2icfixed(test_double[1],
        icSigS15Fixed16ArrayType);
attrValue.val.icXYZ.data[0].Z =double2icfixed(test_double[2],
        icSigS15Fixed16ArrayType);
rc = KcsSetAttribute(profileid, icSigRedColorantTag, &attrValue);
if(rc != KCS_SUCCESS) {
    KcsGetLastError(&errDesc);
    fprintf(stderr, "unable to set red primaries: %s\n", errDesc.desc);
    KcsFreeProfile(profileid);
    return(-1);
}
```

KcsSetCallback()

```
KcsStatusId
KcsSetCallback (KcsFunction function,
        KcsCallbackFunction callback, void *userDefinedData)
```

Purpose

Use `KcsSetCallback()` to associate a callback function with any set of API functions that support callbacks. Those functions are listed in `KcsFunction` (see page 38). If `KcsSetCallback()` is not called for particular values of `KcsFunction`, no callback is issued.

This function allocates resources. To release those resources, set all callback functions to NULL:

```
KcsSetCallback(KcsAllFunc, NULL, NULL);
```

Arguments

`function`

A set of API functions. See "KcsFunction" on page 38 for the list of functions.

`callback`

The application-supplied function to be called when the variable `function` needs to report progress.

`userDefinedData`

Any user-defined data.

Returns

`KCS_SUCCESS`
`KCS_MEM_ALLOC_ERROR`

Example

Code Example 4-12 `KcsSetCallback()`

```
/* template function declaration */

int myProgressCallback(KcsProfileId profileid, unsigned long
        current, unsigned long total, KcsFunction
        operation, void *userDefinedData);

KcsProfileId   completeProfile;
```

Code Example 4-12 `KcsSetCallback()` *(Continued)*

```
KcsPixelLayout pixelLayoutIn;

/* the profiles have been loaded and connected, now set up the
* callback to be active for both the optimize and evaluate
* functions */

status = KcsSetCallback(KcsOptFunc + KcsEvalFunc,
        (KcsCallbackFunction)myProgressCallback, NULL );
if (status != KCS_SUCCESS) {
    fprintf(stderr, "Callback function call failed\n");
}

printf("Optimizing the complete profile \n");
status = KcsOptimizeProfile(completeProfile, KcsOptSpeed, KcsLoadAllNow);
/* check status here*/
/* set up the pixel layout */
status = KcsEvaluate(completeProfile, op, &pixelLayoutIn, &pixelLayoutIn);
/* check status here*/

/* This is my callback function */

int myProgressCallback(KcsProfileId profileid, unsigned long current,
        unsigned long total, KcsFunction operation, void *userDefinedData)

{
    int     pcent;

    pcent = (int) (((float)current/ (float)total) *100.0);
    fprintf(stderr,"Optimize+Evaluate is %3d percent complete\n", pcent);
    fflush(stderr);
    return(KCS_SUCCESS);
/* Free callback resources*/
KcsSetCallback (KcsOptFunc+KcsEvalFunc, NULL, NULL);
}
```

KcsUpdateProfile()

```
KcsStatusId
KcsUpdateProfile(KcsProfileId profile,
        KcsCharacterizationData *charact,
        KcsCalibrationData *calib, void *CMMSpecificData)
```

Purpose

Use `KcsUpdateProfile()` to change the profile data in the loaded profile according to the supplied measurement data.

The data supplied to this call depends on the type of device the profile represents. The default CMM currently supports scanners and monitors; printer profiles are *not* currently supported. The C AP also will be used for printers, when implemented by the default or alternative CMMs. The data required for this call depends on whether the profile is *calibrated* or *characterized*.

Characterization refers to defining the generic color response of all devices of the same make and model (normally by making measurements on a number of sample devices to find an average response). Characterization requires colorimetric measurements. Code Example 4-13 on page 95 shows how these measurements are used to update a profile.

Calibration refers to fine-tuning a specific device's color response. 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.

Arguments

`profile`

The identifier of the profile to be updated.

`*charact`

A set of color sample measurements where sample is a color patch on a test target.

For a scanner, this is a target that is scanned. In this case, each sample measurement consists of an input that is the CIE XYZ value of the patch, as measured. The sample output is the RGB value that the scanner produced when scanning the patch. In

addition, each color sample contains fields for the sample weight, standard deviation, and sample type. The weight is a hint indicating the importance of the sample color. The default should equal 1.0. The standard deviation is used to indicate the statistics of a set of measurements of the sample color that have been reduced to a single sample. The sample type is used to indicate that a color sample represents either black, white, other, neutral, or chromatic color. For best results, the sample type field should be correctly set for each color sample. For example, the `KcsFluorescent` sample type can be used to tag special color samples with this property. The sample type is a hint passed by the KCMS framework to the CMM.

Note that CIE XYZ values are to be scaled in the range `0.0` to `100.0` and that RGB values are to be scaled in range `0.0` to `1.0`. For additional details, see "KcsCharacterizationData" on page 34.

Monitors do not use the `charact` argument. Pointer `*charact` should be set to `NULL` when `KcsUpdateProfile()` is called for a monitor profile. For a monitor, characterization data consists of the profile attributes `icSigRedColorantTag`, `icSigGreenColorantTag`, `icSigBlueColorantTag` and `icSigMediaWhitePointTag`. These attributes must be set and valid prior to calling `KcsUpdateProfile()`. Use `KcsSetAttribute()` to set these attributes.

`*calib`

The linearization tables needed to calibrate the profile. These tables are required to calibrate all device types. They are also required when calling `KcsUpdateProfile()` to characterize a scanner or monitor. Both the input and output spaces are `KcsRGB` for a scanner and monitor. The RGB samples are scaled in the range of `0.0` to `1.0`.

`*CMMSpecificData`

A pointer to any additional data needed by a specific CMM to update the profile. Refer to the CMM documentation for any specific data required. For use with the default CMM, set this argument to `NULL`.

Returns

KCS_SUCCESS
KCS_MEM_ALLOC_ERROR
KCS_CC_UPDATE_NEEDS_MORE_DATA
KCS_CC_UPDATE_INVALID_DATA

Example

To call `KcsUpdateProfile()` successfully, the profile must contain a small number of attributes that identify the type of device the profile represents. It is assumed that the profile already contains these attributes.

An example is given of how to allocate and fill out the arguments required to call `KcsUpdateProfile()`.

Code Example 4-13 `KcsUpdateProfile()`

```
#pragma ident "@(#) kcms_update.c"
/* kcs_update.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <math.h>
#include <kcms/kcs.h>
#include <kcms/kcstypes.h>
#include <kcms/kcsattrb.h>

float   Luminance_float_out[3][256];

/* test code to check profile calibration * /
main(int argc, char **argv)
{
KcsCalibrationData     *calData;
KcsProfileDesc         x_desc, desc;
KcsProfileId           profileid;
KcsStatusId            status;
KcsAttributeValue      attrValue;
KcsErrDesc             errDesc;
int         levels = 256, channels = 3;
int         sizemeas, nvalues, i, j;
FILE        *simfile;
float       input_val;
size_t      rc;

/* Read in the measured calibration data from a file */
/* file lum_out should be located in demo directory with this program */
```

Code Example 4-13 `KcsUpdateProfile()` *(Continued)*

```
if ((simfile = fopen("lum_out", "r")) == NULL) {
    fprintf(stderr,"cannot open output luminance file\n");
    exit(1);
    }

for (i=0; i<channels; i++)
    for (j=0; j<levels; j++)
        Luminance_float_out[i][j] = 0.0;
nvalues = levels * channels;
rc = fread(Luminance_float_out, sizeof(float), nvalues, simfile);
fclose(simfile);

/* Fill out the measurement structures */
sizemeas = (int) (sizeof(KcsMeasurementBase) + sizeof(long) + levels);

calData = (KcsCalibrationData *) malloc(sizemeas);

calData->base.countSupplied = levels;
calData->base.numInComp = 3;
calData->base.numOutComp = 3;
calData->base.inputSpace = KcsRGB;
calData->base.outputSpace = KcsRGB;
for (i=0; i< levels; i++) {
    calData->val.patch[i].weight = 1.0;
    calData->val.patch[i].standardDeviation = 0.0;
    calData->val.patch[i].sampleType = KcsChromatic;

    calData->val.patch[i].input[KcsRGB_R] = (float)i/255;
    calData->val.patch[i].input[KcsRGB_G] = (float)i/255;
    calData->val.patch[i].input[KcsRGB_B] = (float)i/255;
    calData->val.patch[i].input[3] = 0.0;

    calData->val.patch[i].output[KcsRGB_R] = Luminance_float_out[0][i];
    calData->val.patch[i].output[KcsRGB_G] = Luminance_float_out[1][i];
    calData->val.patch[i].output[KcsRGB_B] = Luminance_float_out[2][i];
    calData->val.patch[i].output[3] = 0.0;
    }

calData->val.patch[0].sampleType = KcsBlack;
calData->val.patch[255].sampleType = KcsWhite;

if (!argv[1]) {
    fprintf(stderr, "Usage kcms_update profile_in [profile_out]\n");
```

Code Example 4-13 `KcsUpdateProfile()` *(Continued)*

```
    exit(1);
    }
 /* Let the library open the file */
x_desc.type = KcsSolarisProfile;
x_desc.desc.solarisFile.fileName = argv[optind];
x_desc.desc.solarisFile.hostName = NULL;
x_desc.desc.solarisFile.oflag = O_RDWR;
x_desc.desc.solarisFile.mode = 0;

status = KcsLoadProfile(&profileid, &x_desc, KcsLoadAllNow);
if(status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    printf("LoadProfile error: %s\n", errDesc.desc);
    }

status = KcsUpdateProfile(profileid, NULL, calData, NULL);
if(status != KCS_SUCCESS) {
    status = KcsGetLastError(&errDesc);
    printf("UpdateProfile error: %s\n", errDesc.desc);
    KcsFreeProfile(profileid);
    exit(1);
    }

if (argv[2]) {
    /* Save to an output file */
    desc.type = KcsSolarisProfile;
    desc.desc.solarisFile.fileName = argv[2];
    desc.desc.solarisFile.hostName = NULL;
    desc.desc.solarisFile.oflag = O_RDWR|O_CREAT|O_TRUNC;
    desc.desc.solarisFile.mode = 0666;
    status = KcsSaveProfile(profileid, &desc);
    if(status != KCS_SUCCESS) {
        status = KcsGetLastError(&errDesc);
        printf("SaveProfile error: %s\n", errDesc.desc);
        }
    KcsFreeProfile(profileid);

    } else {
        /* Just save to the same description */
        status = KcsSaveProfile(profileid, NULL);
        if(status != KCS_SUCCESS) {
            status = KcsGetLastError(&errDesc);
            printf("SaveProfile error: %s\n", errDesc.desc);
        }
```

Code Example 4-13 `KcsUpdateProfile()` *(Continued)*

```
    KcsFreeProfile(profileid);
    }
exit(0);
}
```