KCMS Test Suite User's Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 805-3930–10
October 1998

Contents

Preface

The*KCMS Test*™*Suite User's Guide* explains how to test a Kodak Color Management System (KCMS™) color management module (CMM) to verify whether or not the CMM adheres to the KCMS framework. This guide describes a suite of test scripts and the testing facility the CMM developer can use to ensure that a CMM is KCMS-framework compliant. It is a supplemental DDK book in the KCMS documentation.

Who Should Use This Book

This guide is particularly useful if you are a CMM developer. It describes how you can test whether the CMM you have written adheres to the KCMS framework. It is also a reference to anyone interested in the development and use of the KCMS framework.

Typically you would use the test scripts described in this guide to test a CMM you have written for adherence to the framework. This guide assumes you have installed your CMM and its associated profiles. It describes the tests you get with the DDK and how you run them. If you need to change scripts to meet special requirements of your CMM, the guide explains how the script contents are organized. From this information, you can determine what changes you can make. For details on how to use the KCMS test suite in the development of your CMM, see Chapter 6 .

Note - The KCMS test suite can only test the profile attributes it knows about. It is not designed to test new attributes your CMM might add. For details on the supported profile attributes, see the *KCMS Application Developer's Guide.*

Before You Read This Book

Before you read this guide, you should be thoroughly versed in the KCMS framework and in how to write or customize CMMs. This guide *assumes* that you have read the *KCMS Application Developer's Guide*.

In addition, you should have read the following books:

- *KCMS CMM Developer's Guide*
- *KCMS CMM Reference Manual*

All assumptions of the readers of the above books apply to the reader of this guide. To recapitulate key requirements, you should

- Understand C++ and C language
- Be familiar with Solaris dynamic loading technology and all of the associated manual pages
- Understand color science concepts

You should also be familiar with the following manual pages:

- `auto-kcmstest(1)`
- `auto-kcmstest-root(1)`
- kcms_calibrate(1)
- kcms_configure(1)
- `kcms_server(1)`
- `kcmstest(1)`
- `kcms-testreport(1)`

See the on-line `SUNWrdm` packages for information on bugs and issues, engineering news, and patches. For Solaris installation bugs and for late breaking bugs, news, and patch information, see the *Solaris 7 (SPARC Platform Edition) Installation Library* and the *Solaris 7 (Intel Platform Edition) Installation Library* manuals.

For SPARC™ systems, consult the updates your hardware manufacturer may have provided.

How This Book Is Organized

This guide is organized as follows:

Chapter 1 ," summarizes the KCMS test suite. The chapter provides an overview of how the test suite works, it presents the test suite directory hierarchy, and it explains

the approach used to test the KCMS framework so that you know what you can expect from the tests.

Chapter 2 ," gets you started using the `kcmstest` utility, identifies each of the test script commands, and provides the basic script command format. It also describes automated scripts that run several tests once and suggests a scenario for their use.

Chapter 3 ," provides the syntax and a description of each script command keyword.

Chapter 4 ," summarizes the functionality of each test script provided with the DDK.

Chapter 5 ," provides an annotated script example showing how to set each supported attribute.

Chapter 6 ," threads together the procedure for using this test suite. The chapter provides references to the relevant documentation on developing and testing KCMS CMMs.

Appendix A ," associates status code values and strings.

Glossary "is a list of words and phrases found in this book along with their definitions.

Related Books

The following is a list of recommended books that can help you accomplish the tasks described in this guide:

- *International Color Consortium (ICC) Profile Format Specification* (located on-line in `/opt/SUNWsdk/kcms/doc/icc.ps`). For the most current version of the ICC specification, see the web site at `http://www.color.org`.
- White papers on color science provided with the KCMS product.

Ordering Sun Documents

The SunDocs^{SM} program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals using this program.

For a list of documents and how to order them, see the catalog section of the SunExpress^{TM} Internet site at `http://www.sun.com/sunexpress`.

Note - The term “x86” refers to the Intel 8086 family of microprocessor chips, including Pentium and Pentium Pro processors and compatible microprocessor chips made by AMD and Cyrix. In this document, the term “x86” refers to the overall platform architecture, whereas “*Intel Platform Edition*” appears in the product name.

What Typographic Changes Mean

The following table describes the typographic changes used in this guide.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename.*
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User’s Guide*. These are called *class* options. You *must* be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

KCMS Test Suite Overview

In This Chapter

This chapter explains what the KCMS test suite is, summarizes how it works, and provides the testing approach so you know what to expect from the tests. For information on the KCMS development environment, see “Development Environment Requirements” on page 77 .

What is the KCMS Test Suite?

The KCMS test suite is a set of scripts that test the KCMS “C” application program interface (API). The KCMS “C” API is described in detail in the *KCMS Application Developer’s Guide.*

In addition to enhancement scripts that support new features and fix bugs, the KCMS test suite includes one or more scripts that correspond to each of the following KCMS functions:

- `KcsLoadProfile()()`
- `KcsConnectProfile()()`
- `KcsEvaluateProfile()()`
- `KcsOptimizeProfile()()`
- `KcsModifyLoadHints()()`
- `KcsSaveProfile()()`

- `KcsGetAttribute()`
- `KcsSetAttribute()()`
- `KcsUpdateProfile()()`

All the KCMS test scripts contain commands. In general, a script *command* corresponds to each KCMS "C"API function call. The test scripts organize the commands to be executed according to the guidelines in the *KCMS Application Developer's Guide*. A few commands that accept variable-length input vary slightly from the API structure. See Chapter 5 ," for details on these exceptions.

All the test scripts in each functional category perform operations to confirm that subsequent KCMS API functions such as connecting profiles and evaluating the results can be performed. At the conclusion of each test, the profile(s) are freed.

Some additional commands in the KCMS test suite facilitate scripting and reading of a log file that contains the test results.

How the Test Suite Works

You use `kcmstest`, a script-driven utility that you run from a command shell, to test your CMM for KCMS framework interaction. `kcmstest` is supported on SPARC™ and x86 platforms.

`kcmstest` interprets each script command, and the corresponding KCMS framework function call is performed. Then the next script command is read and again the appropriate framework function call is made. Any data or information that needs to be maintained to make the sequence of function calls coherent is provided by `kcmstest`.

Various options to `kcmstest` allow you to run one to several test scripts. As each test script command executes, information about it is displayed to the command shell window and to a log file. If at any time during execution of a KCMS framework function call an unexpected status is returned, the test is immediately aborted.

Approach To Testing

The KCMS test scripts are organized to focus on a specific function call and exercise it through the range of its parameters. Because some functions depend upon the successful completion of previous functions, by necessity, a given test consists of several different API function calls.

To absolutely verify that a profile is loaded successfully would require examining internal framework variables for specific values. Such an approach to testing the API is too intrusive to be effective. The KCMS test scripts, instead, rely on the status returned from each of the KCMS API functions along with some inferred conclusions about the results of functions yet to be executed. For example, the status returned from connecting two profiles is one indication that a connection succeeded. Following this, the new complete profile can be used in a call to `KcsEvaluate()()` and the status returned from the evaluation can be used as another indication of the success of the connection. This assumes that the evaluation has no errors associated with it. If you want to further verify the connection, you can examine the image resulting from the call to `KcsEvaluate()()` and compare it to some expected output.

In the above testing scenario, subsequent framework calls are used to verify an initial call, and conclusions about the initial call are drawn from the results of subsequent calls.

The ultimate goal of using the KCMS framework is to evaluate the results of applying color correction to images. The test images are organized in TIFF file format. To preserve system resources, many of the test scripts do not save the resulting TIFF image (however, you have the option to save the image). The main test concern is to demonstrate that the evaluation completes successfully for a given profile.

The scripts described in this guide do not focus on the color quality of the images tested. In a few cases, the color-managed image can be displayed for verification purposes, however the primary focus of the tests is to demonstrate the software color quality. In most cases, you must visually inspect an image to verify it.

Extending Testing For Your CMM

The existing profiles use the default CMM provided with KCMS. To extend this testing for your own CMM and resulting profiles, you may choose to replace profile names in some of the tests with your own similar profiles. (That is, replace a monitor profile with your own monitor profile, a scanner profile with your scanner profile, and printer profile with your printer profile.) The CMM Id in the profile will cause your CMM to be loaded for the resulting tests. Instead of modifying existing scripts for your profiles, you may choose to create new ones. In the same manner as profiles, data for updating your profiles may be replaced with your own data, and images may be replaced with your own TIFF file images.

CHAPTER 2

Running KCMS Test Scripts

In This Chapter

This chapter explains the basic information you need to run the KCMS test scripts. It describes the file hierarchy of the testing environment, introduces you to the script commands, and shows the basic script command format. Finally it provides two methods of running the test scripts: one using the `kcmstest` command and a second, using automated script files.

Getting Started

Packaging

To run the KCMS test suite, you first must install the Solaris operating system. It includes the KCMS Software Development Kit (SDK) package, which contains the KCMS “C” API functions.

The KCMS test suite is a packaged in the KCMS Driver Development Kit (DDK). When you package add the DDK, the test suite files are installed in the `/opt/SUNWddk/kcms/kcmstest` directory.

Environment Variables

To run the scripts, you need to know about two environment variables: `KCMSROOT` and `KCMS_PROFILES`.

`KCMSROOT` specifies the path to the top of the `kcmstest` directory.

`KCMS_PROFILES` specifies the path to the `kcmstest/profiles` directory. See Figure 2–1 .

Prior to running test scripts using the `kcmstest` command, you set these variables from the command line, for example

%**`setenv KCMSROOT`***path*

where *path* is the path to the `kcmstest` directory.

Alternately, if you run the automated script files, you set the variables at the time you run the scripts. See “Using Automated Script Files To Run Test Scripts” on page 14 for details.

Required File Hierarchy

Figure 2–1 shows the required directory structure you need to run test scripts. When you package add the test suite, the `kcmstest` directory contains the structure shown in the figure.

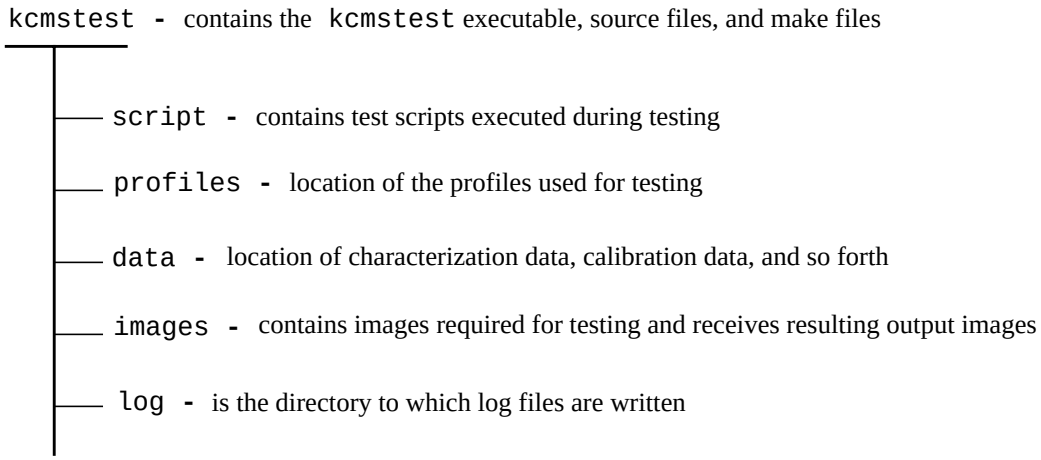

Figure 2–1 `kcmstest` File Hierarchy

`kcmstest` Directory

The `kcmstest` directory is at the top of the test suite hierarchy in Figure 2–1 . It contains the executables necessary to run the test suite.

The initialization file `icc.ini` in this directory lists the all the default test scripts that are packaged with the test suite. See “Initialization File” on page 7 for details on the contents of this file.

Significant Directories

Four directories shown in Figure 2–1 are of particular significance. These are

- `script`
- `profiles`
- `data`
- `log`

The `script` directory contains the test scripts to be executed. By default, this directory includes all the test scripts listed in `icc.ini`. You can run a subset of the scripts, or specify an alternate initialization file when you run the `kcmstest` command. See “Using `kcmstest` To Run Test Scripts” on page 10 for details. If you have written customized versions of scripts to test your CMM, you must install them in this directory.

The `profiles` directory contains a default set of profiles used with the default test scripts. You can install the profiles used by your CMM into this directory. Note that this is a separate installation from the one you do to make your CMM profiles available to the KCMS framework. For details, see Chapter 6 .

The `data` directory contains measurement and calibration data.

The `log` directory contains output. This directory initially is empty. It holds the results of running test scripts.

Images

The `images` directory contains images resulting from running the test suite and test TIFF images.

Initialization File

The default initialization file `icc.ini` is shown in Code Example 2–1 .

CODE EXAMPLE 2–1 Initialization file `icc.ini`

```
[Verbose] [ProfilePath]
profiles/ [ImagePath] images/ [DataPath] data/ [NumberOfTests] 30 [Tests]

IC_lhints.scr IC_conerr.scr IC_lana.scr IC_eval.scr IC_lmany.scr
```

```
IC_optspeed.scr IC_connect.scr IC_evalmany.scr IC_attr1.scr IC_attr2.scr

IC_layouts.scr IC_conmany.scr IC_optsize.scr IC_evalerr.scr IC_update1.scr

IC_update2.scr IC_xprofile.scr IC_xprofilehost.scr IC_xprofilesav.scr

IC_xprofilesavremote.scr IC_xwindow.scr IC_xwindowerr.scr IC_xdisplay.scr

IC_evalplus.scr IC_pacbug.scr IC_loadsol.scr IC_sun_update.scr IC_gray.scr

IC_gamut.scr IC_lut.scr
```

Note - The `icc.ini` file does not include the tests, `IC_xprofilesavroot.scr` and `IC_updatewin.scr`, which must be run as root. To run `IC_xprofilesavremote.scr`, you need to change the `DISPLAY` environment variable. See the comments in the automated test scripts (`auto-kcmstest` and `auto-kcmstest-root`) for details.

The `icc.ini` file contains the path to the profiles, images, and data required to run the test scripts. In addition, it lists the number of test scripts following the `[NumberOfTests]` field, and it lists the filename of each test script.

If your CMM requires a different set of test scripts, you can create an alternate initialization file. Say, for example, you edited several of the scripts to test special features of your CMM. In such a case you need to install the scripts you plan to test with in the `script` directory. To add to the existing initialization file, you also must create an alternate file that reflects test script changes. See “Creating An Alternate Initialization File” on page 8 ,” for details.

Creating An Alternate Initialization File

You can create an alternate initialization file if, for example, you customized scripts for your CMM.

To create the file (see Code Example 2–1),

1. **Use a text editor to save a copy of** `icc.ini` **under a new filename, for example** `alternate.ini`**.**

2. **Add (or remove) test script name(s) in the file list.**

3. **Change the value immediately following the** `[NumberOfTests]` **field to update the number of tests.**

KCMS Test Script Commands

In general, a KCMS test script command corresponds to each of the KCMS “C” API functions. Additionally, there are some commands that are necessary to facilitate scripting and reading the test results log. See Chapter 3 ,” for a detailed description of each command. Table 2–1 lists each of the script commands and the KCMS “C” API function to which it corresponds.

TABLE 2–1 Test Script Commands and “C” API Functions

Test Script Command	KCMS “C” API Function
CONNECT:	KcsConnectProfile()()
CREATE:	KcsCreateProfile()()
EVAL:	KcsEvaluate()()
FREE:	KcsFreeProfile()()
GETATTR:	KcsGetAttribute()()
LOAD:	KcsLoadProfile()()
LOG:	No specific function. It writes to a log file.
MODIFYLH:	KcsModifyLoadHints()()
OPTIMIZE:	KcsOptimizeProfile()()
SAVE:	KcsSaveProfile()()
SETATTR:	KcsSetAttribute()()
UPDATE:	KcsUpdateProfile()()

Script Command Format

A single script command consists of the command name (including the colon), followed by one or more keyword/value pairs. A keyword is separated from its value by an equal sign (=). Each keyword/value pair ends with a semicolon(;).

The basic script command format is shown below:

```
COMMAND_NAME:keyword=value; keyword=value;
```

You can free-format test scripts. That is, you can insert any whitespace character into any script command.

Code Example 2–2 shows an actual test script that demonstrates some of the script commands and their associated keywords and values.

CODE EXAMPLE 2–2 Sample Test Script Showing Commands

```
LOAD:Reference=scanner;
Profile=mtk600zs.inp; Handling=File; LoadHint=AllNow; LOAD:Reference=monitor;

Profile=sony16.mon; Handling=File; LoadHint=AllNow; CONNECT:NAME=scan-mon;

 Count=2;      Reference=scanner;      Reference=monitor;

Operation=FORWARD; EVAL:Reference=scan-mon;

SourcePixLayout=RGBInterLeaved;      DestPixLayout=RGBInterLeaved;

Callbacks=;      ImageIn=rhg_mtek600;      ImageOut=rhg_mon.tst;

Operation=Forward; FREE:Reference=scanner; FREE:Reference=monitor;

FREE:Reference=scan-mon;
```

Using `kcmstest` To Run Test Scripts

The `kcmstest` command is a test script interpreter that reads test scripts and performs the KCMS “C” API function calls based upon the commands in the test script.

To run test scripts with this command, you use the procedure described below. For details on `kcmstest`, see the manual page.

Starting the kcmstest Command

Note - Be sure to set the KCMSROOT environment variable before using the kcmstest command. See “Environment Variables” on page 6 for details.

The simplest way to start kcmstest is to type the following from a command shell and press **Return**.

```
%kcmstest
```

You are prompted with the following message:

```
Enter
the script name to be executed or ‘‘quit’’ to exit Script

Name(s)?
```

You can enter the name of a script, for example IC_attrl.scr. Alternately, you can enter **all**, which executes all the scripts listed in icc.ini.

Note - You must perform a few tasks manually to be able to run all the test scripts when you enter **all**. See the contents of the auto-kcmstest script for details.

When you run individual test scripts, an output log file is generated for each script. When you run all the scripts listed in icc.ini, a single log file is generated. See “Recording Test Script Results To a Log File” on page 12 for details.

Note - Use the test script auto-kcmstest to run the entire icc.ini test list. (See “Using Automated Script Files To Run Test Scripts” on page 14 .) The script creates certain setup files automatically.

Command Line Options -i, -h, -s

From the command line, you can enter various options to the kcmstest command. Three frequently used options are –i, –h, and –s.

To specify your own initialization file, you can enter its name on the command line preceded by the –i option, for example

```
%kcmstest -i
optional.ini
```

See “Creating An Alternate Initialization File” on page 8 for details on alternate initialization files.

You can use the –s option to specify a script name (or **all**) and the –h option, to specify an alternate legal remote host name for scripts that test remote host access.

The –h option attempts to pull a profile from the default directories on the remote host. Be sure that host has these directories and profiles.

The following example specifies the alternate initialization file `alternate.ini`, the script `IC_attrl.scr`, and the alternate host name `dusk`:

```
%kcmstest -i
alternate.ini -s IC_attrl.scr -h dusk
```

The example below defaults to using the `icc.ini` file:

```
%kcmstest -s
all
```

In this example, if any of the scripts in the `icc.ini` file access a remote host, the host name will be `NULL` and the scripts will fail.

Script Display

As each of the test script commands is executed, information about the command that is currently being interpreted is displayed to the command shell window as well as written to a log file in the `kcmstest/log` directory.

Recording Test Script Results To a Log File

For each script file executed, results are recorded in a log file. All the log files can be found in the `kcmstest/log` directory. The log file name is the name of the script file, with the `.scr` file extension replaced by the `.log` extension. If, for example, the test script name is `IC_eval.scr`, the log file name is `IC_eval.log`.

One exception to this naming scheme is if you enter **`all`** as the test script name. See “Starting the `kcmstest` Command” on page 11 for details on this entry. In this case, the log file name is `testall.log`.

Two versions of a log file may exist at any given time: the current and the previous version. The previous version has its extension changed to `.bak`.

Code Example 2–3 is the log file output created from the test script shown in Code Example 2–2 .

CODE EXAMPLE 2–3 Log File Output

```
Parsing a KcsLoadProfile
Command         Profile Reference = scanner         Profile File Name =

kcmsEKls3510.inp         Profile Handling = By File         Profile Load Hint

= LoadWhenNeeded;         Profile Load Hint = UnLoadwhenNeeded;

Profile Operation Hint = Image;         Load Hint =   2024000         Thu Jul
```

```
25 08:16:07 1996  Completed KcsLoadProfile command, status =    0         Thu
Jul 25 08:16:07 1996    Parsing a KcsLoadProfile Command         Profile
Reference = printer         Profile File Name = kcmsEKsunnws.out
Profile Handling = By File         Profile Load Hint = LoadWhenNeeded;
Profile Load Hint = UnLoadwhenNeeded;         Profile Operation Hint = Image;
       Load Hint =   2024000         Thu Jul 25 08:16:07 1996  Completed
KcsLoadProfile command, status =    0         Thu Jul 25 08:16:07 1996
Parsing a KcsConnectProfiles Command         Profile Reference = scan-print
     Number of Profiles in Connect = 2         Profile Reference = scanner
    Profile Reference = printer         Operation Hint =     20001         Thu
Jul 25 08:16:07 1996  Completed KcsConnectProfiles command, status =    0
   Thu Jul 25 08:16:08 1996    Parsing a KcsEvaluate Command         Profile
Reference = scan-print         Source Layout = RGBInterLeaved;
Destination Layout = RGBInterLeaved;         Input Image Name =
macbeth_1550.tif         Output Image Name = None         Operation Hint =
20001         Thu Jul 25 08:16:08 1996  Completed KcsEvaluate command, status
=    0         364800.000000 pixels processed in 0.621338 seconds.
The processing rate = 587120.062500 pixels/second.   Parsing a Free Profile
command          Profile reference =scanner  Completed KcsFreeProfile command,
status =    0  Parsing a Free Profile command          Profile reference
=printer  Completed KcsFreeProfile command, status =    0  Parsing a Free
Profile command          Profile reference =scan-print  Completed
KcsFreeProfile command, status =    0
```

Status Codes

If at any time during script execution, a KCMS framework API function call returns with an unexpected status code, the test is immediately aborted. For a list of all the status codes strings and their values, see Appendix A .

Note - It may be your intention to have a status code returned that indicates an error because you deliberately set up a script to test an error condition. The script commands provide the optional keyword `XStatus`, which allows you to do this. For details, see the script command descriptions in Chapter 3 . Also see “Checking Status Codes” on page 79 .

Using Automated Script Files To Run Test Scripts

The `kcmstest` directory includes two automated scripts: `auto-kcmstest` and `auto-kcmstest-root`.

Note - See “Tips on Running the Automated Test Scripts” on page 15 before using this testing method.

Using `auto-kcmstest`

The `auto-kcmstest` script allows you to run the complete test suite in `icc.ini`, including scripts in the `icc.ini` file list that access a remote host. This script is located in the `kcmstest` directory.

Note - You may need to edit the script to change path information.

To run this script, do not set the environment variable `KCMSROOT` with the `setenv` command. Instead, provide two arguments: the `KCMSROOT` environment variable as the first argument and the remote host name as the second, for example

```
%auto-kcmstest
/opt/SUNWddk/kcms dusk
```

In this example, `/opt/SUNWddk/kcms` is the `KCMSROOT` environment variable and `dusk` is the remote host name. Note that if you are in the directory where `auto-kcmstest()` is located, only the host argument is needed, for example

```
%auto-kcmstest
dusk
```

Using auto-kcmstest-root

Certain test scripts require that you be root to run them. You would use these tests if, for example, you wanted to create an X Window System profile in a root-owned directory. To run these scripts, a second automated script called auto-kcmstest-root is provided.

To run the auto-kcmstest-root script,

1. **Become superuser.**

```
%su
```

2. **Provide one argument: the KCMSROOT environment variable, for example**

```
#./auto-kcmstest-root /opt/SUNWddk/kcms
```

   Note that if you are in the directory containing auto-kcmstest-root, no argument is required.

Getting a Failure and Performance Report

After you have run the complete test suite using auto-kcmstest and auto-kcmstest-root, you can get an automated failure and performance report by running the kcms-testreport command. This command takes two arguments: the name of the test log and the report title. Very likely, you would redirect output to a file of the same name as the report title, for example

```
%kcms-testreport
log/testall.log my_test_1 > my_test_2
```

In this example, my_test_1 is the report title and my_test_2 is the output filename.

Tips on Running the Automated Test Scripts

The following is a suggested sequence for running a complete test suite using the automated script files:

1. **Run the auto-kcmstest script, for example**

```
%auto-kcmstest
   /opt/SUNWddk/kcms dusk
```

2. **Become root, for example**

```
%su
```

3. **Run the** `auto-kcmstest-root` **script, for example**

```
#./auto-kcmstest-root
   /opt/SUNWddk/kcms
```

4. **Run** `kcms-testreport` **and redirect output to a file, for example**

```
#kcms_testreport
   log/testall.log my_test_1 > my_test_2
```

Note - `auto-kcmstest-root` must be run after `auto-kcmstest` because it appends its resulting logs to the `auto-kcmstest` log file.

You may want to redirect the automated-test-script output to a file, as it is quite lengthy.

CHAPTER 3

KCMS Test Suite Commands

In This Chapter

This chapter alphabetically presents each of the `kcmstest` test script commands. For each command, the chapter provides a summary description, the command syntax, and a detailed description of each keyword. Generally, all of the command keywords must be used for a command to execute successfully. The text indicates when certain keywords do not need to be used.

Table 3–1 lists each of the test script commands and the KCMS "C" API function call to which it corresponds.

TABLE 3–1 Test Script Commands and "C" API Functions

Test Script Command	KCMS "C" API Function
`CONNECT:`	`KcsConnectProfile()()`
`CREATE:`	`KcsCreateProfile()()`
`EVAL:`	`KcsEvaluate()()`
`FREE:`	`KcsFreeProfile()()`
`GETATTR:`	`KcsGetAttribute()()`

TABLE 3–1 Test Script Commands and "C" API Functions *(continued)*

Test Script Command	KCMS "C" API Function
LOAD:	KcsLoadProfile()()
LOG:	No specific function. It writes to a log file.
MODIFYLH:	KcsModifyLoadHints()()
OPTIMIZE:	KcsOptimizeProfile()()
SAVE:	KcsSaveProfile()()
SETATTR:	KcsSetAttribute()()
UPDATE:	KcsUpdateProfile()()

CONNECT:

CONNECT: Command Description

CONNECT: functionality corresponds to the KcsConnectProfile()() call. When this command is command is interpreted, the KcsConnectProfile()() function is executed and the status is reported back from the kcmstest command display to the test log.

CONNECT: Command Syntax Example

```
CONNECT:NAME=reverse; Count=2; Reference=monitor;
Reference=scanner; Operation=Forward;
```

CONNECT: Keywords and Values

Table 3–2 presents the CONNECT: command keywords and their descriptions.

TABLE 3–2 CONNECT: Command Keywords

Keyword	Description
Name=	Is the reference name that will be assigned to the new profile if the CONNECT:command completes successfully.
Reference=	Is the name that was assigned when the profiles were loaded or created.
Count=	Is the number of profiles that will be used in the connection. Currently two profiles are used to connect forward and reverse profiles. Three profiles are used to connect simulate profiles.
Operation=	Defines the operation load hint that will be used to connect the new profile. This keyword has the operation and content hint values shown in Table 3–3 . It indicates what transforms in the profiles will be loaded and connected in the final (complete) profile. An Operation= keyword can appear more than once in a single script command. Multiple operations are logically OR'd together.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non- success status, the XStatus= is followed by the corresponding expected, non-success status in hexadecimal format.

Table 3–3 shows the acceptable values for the Operation= keyword.

TABLE 3–3 CONNECT: Command Operation= Keyword Values

Value	Load Hint Set
Forward;	KcsOpForward
Reverse;	KcsOpReverse
Simulate;	KcsOpSimulate
Gamut;	KcsOpGamutTest
OpsAll;	KcsOpAll

TABLE 3–3 CONNECT: Command Operation= Keyword Values *(continued)*

Value	Load Hint Set
ContUnkn;	KcsContUnknown
Graphics;	KcsContGraphics
Image;	KcsContImage
ColorMtrc;	KcsContColorimetric
ContAll;	KcsContAll

CREATE:

CREATE: Command Description

CREATE: functionality corresponds to the KcsCreateProfile()( ) call. When this command is interpreted, the KcsCreateProfile()( ) function is executed and the status is reported back from the kcmstest command display to the test log. The CREATE: command creates a generic profile with the default CMM Id.

CREATE: Command Syntax Example

```
CREATE:Reference=umax;
```

CREATE: Keywords and Values

Table 3–4 presents the CREATE: command keywords and their descriptions.

TABLE 3–4 CREATE: Command Keywords

Keyword	Description
Reference=	Is the name that this profile will be referred to in subsequent script commands. In the context of the test script, it is the profile name.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus keyword is not required. In cases where a script command is expected to return a non-success status, the keyword is followed by the corresponding expected, non-success status in hexadecimal format.

EVAL:

EVAL: Command Description

EVAL: functionality corresponds to the KcsEvaluate()() call. When this command is interpreted, the KcsEvaluate()() function is executed and the status is reported back from the kcmstest command display to the test log.

EVAL: Command Syntax Example

```
EVAL:Reference=forward; SourcePixLayout=RGBInterLeaved;
DestPixLayout=RGBInterLeaved; Callbacks=;
ImageIn=test; ImageOut=None; Operation=Forward;
```

EVAL: Keywords and Values

Table 3–5 presents the EVAL: command keywords and their descriptions.

TABLE 3–5 EVAL: Command Keywords

Keyword	Description
`Reference=`	Is the name that was assigned when the profiles were connected.
`SourcePixLayout=`	Sets the pixel layout structure and, if necessary, restructures the input data. This keyword has one of the following values: `RGBInterLeaved` (also called component- or pixel-interleaved) `RGBPlanar` `RGBRowInterleaved` (also called planar- or band-interleaved) (For details on these values, see the description of the KcsPixelLayout structure in the SDK manual *KCMS Application Developer's Guide*)

`DestPixLayout=`	Sets the pixel layout structure. This keyword has one of the following values:
	`RGBInterLeaved`
	`RGBPlanar`
	`RGBRowInterleaved`

`ImageIn=`	Is the image file name that will be processed in the `EVAL:` command. The image file must be located in the `kcmstest/images` directory. Only images stored in the TIFF file format can be processed by `kcmstest` at this time.
`ImageOut=`	Is the image file name that will be output from the `EVAL:` command. The image file will be located in the `kcmstest/images` directory. Only TIFF image file format can be output by `kcmstest` at this time. In the event that no image output is required, specify `None` for the image name. Note that TIFF files can use up your disk space very quickly. Be sure to remove them after inspection. Specify `NULL` if you do not want to save the output image.
`Operation=`	Defines the operation load hint that will be used to evaluate the data. This keyword has the operation and content hint values listed inTable 3–6 . In the table, the keyword value is followed by the corresponding value set in the `Operations` parameter passed to the `KcsEvaluate()` function. Only one direction and one content operation hint can appear in a single `EVAL:` script command, and the complete profile that is evaluated must include the matching operation hint (transform). If, for example, you connect profiles requesting the `Reverse` operation, and you evaluate the resulting profile requesting the `Forward` operation, you will get an error.

TABLE 3–5 EVAL: Command Keywords *(continued)*

Keyword	Description
Callbacks=	Causes kcmstest to perform a KcsSetCallback()( ) call. Callbacks are registered in the log file.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non-success status, the keyword is followed by the corresponding expected, non-success status in hexadecimal format.

Table 3–6 presents the acceptable values for the EVAL: command Operation= keyword.

TABLE 3–6 EVAL: Command Operation= Keyword Values

Value	Value Set
Forward;	KcsOpForward
Reverse;	KcsOpReverse
Simulate;	KcsOpSimilate
Gamut;	KcsOpGamutTest
ContUnkn;	KcsContUnknown
Graphics;	KcsContGraphics
Image;	KcsContImage
ColorMtrc;	KcsContColorimetric

Note - The EVAL: command also produces pixel evaluation speeds, in terms of 24-bit pixels per second, for the log file.

FREE:

FREE: Command Description

FREE: functionality corresponds to the KcsFreeProfile()() call. When this command is interpreted, the KcsFreeProfile()() function is executed and the status is reported back from the kcmstest command display to the test log.

FREE: Command Syntax Example

```
FREE:Reference=scanner;
```

FREE: Keywords and Values

Table 3–7 presents the FREE: command keywords and their descriptions.

TABLE 3–7 FREE: Command Keywords

Keyword	Description
Reference=	Is the name that was assigned to the profile either when it was loaded or created using the CONNECT: or CREATE: command. If the file was loaded with the LOAD: or CREATE: command, the file is closed.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non-success status, the keyword is followed by the corresponding expected, non-success status in hexadecimal format.

GETATTR:

GETATTR: Command Description

GETATTR: functionality corresponds to the KcsGetAttribute()() call. When this command is interpreted, the KcsGetAttribute()() function is executed and the status is reported back from the kcmstest command display to the test log.

GETATTR: Command Syntax Example

```
GETATTR:Reference=testscanner; Attribute Tag=icSigMediaWhitePointTag;
```

GETATTR: Keywords and Values

Note - If all the attributes for a given profile are required, set the keyword Attribute Tag to the value All. This will cause kcmstest to retrieve and display all the attributes and values.

Table 3–8 presents the GETATTR: command keywords and their descriptions.

TABLE 3–8 GETTATTR: Command Keywords

Keyword	Description
Reference=	Is the name that was assigned when the profiles were loaded, created, or connected.
Attribute Tag=	Is the name of the attribute that is to be manipulated by this command. For a list of all the attribute names, see Chapter 5, "KCMS Profile Attributes," in the *KCMS Application Developer's Guide*. The section entitled "List of All Attributes" lists all the attribute names you can use as values for this keyword.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non-success status, the keyword is followed by the corresponding expected, non-success status in hexadecimal format.

LOAD:

LOAD: Command Description

LOAD: functionality corresponds to the KcsLoadProfile()() call. When this command is interpreted, the KcsLoadProfile()() function is executed and the status is reported back from the kcmstest command display to the test log.

LOAD: Command Syntax Example

```
LOAD:Reference=scanner; Profile=clc500fs.inp; Handling=File;
LoadHint=AllNow;
```

LOAD: Keywords and Values

Table 3–9 presents the LOAD: command keywords and their descriptions.

TABLE 3–9 LOAD: Command Keywords

Keyword	Description
Reference=	Is the name that this profile will be referred to in subsequent script commands. In the context of the test script, it is the profile name.
Profile=	Is the file name of the profile. All profiles must be located in the kcmstest/profiles directory. This keyword may name a pre-made profile or a profile that is created as part of the test script.
Handling=	Describes how the profile will be handled. This keyword has the values listed in Table 3–10 .
LoadHint=	Defines the load hint that will be used to load the profile. This keyword has the values listed in Table 3–11 . In the table, the keyword value is followed by the corresponding value set in the loadHints parameter passed to the KcsLoadProfile()() function. A LoadHint= keyword can appear more than once in a single script command. Multiple load hints are logically OR'd together.

TABLE 3–9 LOAD: Command Keywords *(continued)*

Keyword	Description
Operation=	Defines the operation load hint that will be used to load the profile. This keyword has the operation and content hint values listed in Table 3–12 . In the table, the keyword is followed by the corresponding value set in the loadHints parameter passed to the KcsLoadProfile()( ) function. An Operation= keyword can appear more than once in a single script command. Multiple operations are logically OR'd together.
KcsDisplay=	Is the X Window System display (display:screen) number in the form of 0.0, 0.1, and so forth. Use this keyword for multiheaded systems.
KcsHost=	Is the host name of the workstation from which a profile is to be read. You must have the kcms_server(1) daemon running to access another host through the network.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non-success status, the keyword is followed by the corresponding expected, non-success status in hexadecimal format.

Table Table 3–10 presents the LOAD: command Handling= keyword values.

TABLE 3–10 LOAD: Command Handling= Keyword Values

Value	Description
File	Sets Desc.type = KcsFileProfile
KcsSolarisFile	Sets Desc.type = KcsSolarisProfile
KcsWindow	Sets Desc.type = KcsWindowProfile
Memory	Sets Desc.type = KcsMemoryProfile

Table 3–11 presents the LOAD: command LoadHint= keyword values.

TABLE 3–11 LOAD: Command LoadHint= Keyword Values

Value	Load Hint Set
AllNow	KcsLoadAllNow
AllWhen	KcsLoadAllWhenNeeded
LoadAttr	KcsLoadAttributeNow
MinMem	KcsLoadMinimalMemory
PurgeMem	KcsPurgeMemoryNow
LoadWhenNever	KcsLoadNever
LoadWhenNow	KcsLoadNow
LoadWhenNeeded	KcsLoadWhenNeeded
LoadWhenIdle	KcsLoadWhenIdle
UnloadWhenNow	KcsUnloadNow
UnloadWhenFree	KcsUnloadWhenFreed
UnloadWhenNeeded	KcsUnloadWhenNeeded
UnloadAfter	KcsUnloadAfterUse
WhatAttr	KcsAttributes
WhatAll	KcsAll
WhatEffects	KcsEffect

Table 3–12 presents the LOAD: command Operation= keyword values.

TABLE 3–12 LOAD: Command Operation= Keyword Values

Value	Load Hint Set
OpsAll	KcsOpAll
ContUnkn	KcsContUnknown
Graphics	KcsContGraphics
Image	KcsContImage
ColorMtrc	KcsContColorimetric
ContAll	KcsContAll

LOG:

LOG: Command Description

LOG: writes a string to the log file to facilitate reading test results. This command does not correspond to a KCMS function call.

LOG: Command Syntax Example

```
LOG:Connect Test-Connect profiles varying the number of; LOG:profiles.;
```

LOG: Keywords and Values

None.

MODIFYLH:

MODIFYLH: Command Description

MODIFYLH: functionality corresponds to the KcsModifyLoadHints()() call. When this command is interpreted, the KcsModifyLoadHints()() function is executed and the status is reported back from the kcmstest command display to the test log. This command is commonly used when a profile has previously been loaded for attributes only. It allows the rest of the profile to be loaded.

MODIFYLH: Command Syntax Example

```
MODIFYLH:Reference=connected; LoadHint=LoadAllNow;
```

MODIFYLH: Keywords and Values

Table 3–13 presents the MODIFYLH: command keywords and their descriptions.

TABLE 3–13 MODIFYLH: Command Keywords

Keyword	Description
Reference=	Is the name that was assigned when the profiles were loaded, created, or connected.
LoadHint=	Defines the load hint that will be used to load the profile. This keyword has the values shown in Table 3–14 . In the table, the keyword value is followed by the corresponding value set in the loadHints parameter passed to the KcsLoadProfile()() function. A LoadHint= keyword can appear more than once in a single script file. Multiple load hints are logically OR'd together.

TABLE 3–13 MODIFYLH: Command Keywords *(continued)*

Keyword	Description
Operation=	Defines the operation load hint that will be used to connect the new profile. This keyword has the operation and content hint values shown in Table 3–15 . In the table, the keyword value is followed by the corresponding value set in the loadHints parameter passed to the KcsLoadProfile()() function. An Operation= keyword can appear more than once in a single script command. Multiple load hints are logically OR'd together.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non-success status, the keyword is followed by the corresponding expected, non-success status in hexadecimal format.

Table 3–14 presents the MODIFYLH: command LoadHint= keyword values.

TABLE 3–14 MODIFYLH: Command LoadHint= Keyword Values

Value	Load Hint Set
AllNow	KcsLoadAllNow
AllWhen	KcsLoadAllWhenNeeded
LoadAttr	KcsLoadAttributeNow
MinMem	KcsLoadMinimalMemory
PurgeMem	KcsPurgeMemoryNow
LoadWhenNever	KcsLoadNever
LoadWhenNow	KcsLoadNow
LoadWhenNeeded	KcsLoadWhenNeeded
LoadWhenIdle	KcsLoadWhenIdle
UnloadWhenNow	KcsUnloadNow

TABLE 3–14 MODIFYLH: Command LoadHint= Keyword Values *(continued)*

Value	Load Hint Set
UnloadWhenFree	KcsUnloadWhenFreed
UnloadWhenNeeded	KcsUnloadWhenNeeded
UnloadAfter	KcsUnloadAfterUse
WhatAttr	KcsAttributes
WhatAll	KcsAll
WhatEffects	KcsEffect

Table 3–15 presents the MODIFYLH: command Operation= keyword values.

TABLE 3–15 MODIFYLH: Command Operation=Keyword Values

Value	Load Hint Set
OpsAll	KcsOpAll
ContUnkn	KcsContUnknown
Graphics	KcsContGraphics
Image	KcsContImage
ColorMtrc	KcsContColorimetric
ContAll	KcsContAll

OPTIMIZE:

OPTIMIZE: Command Description

OPTIMIZE: functionality corresponds to the KcsOptimizeProfile()() call. When this command is interpreted, the KcsOptimizeProfile()() function is executed and the status is reported back from the –kcmstest command display to the test log.

OPTIMIZE: Command Syntax Example

```
OPTIMIZE:Reference=simulate; Optimization=Speed;
```

OPTIMIZE: Keywords and Values

Table 3–16 presents the OPTIMIZE: command keywords and their descriptions.

TABLE 3–16 OPTIMIZE: Command Keyword Values

Keyword	Description
Reference=	Is the name that was assigned when the profiles were loaded, created, or connected.
Optimization=	Sets the optimization type. This keyword has the values shown in Table 3–17 . In the table, the keyword value is followed by the corresponding value set in the optimizationType parameter passed to the KcsOptimizeProfile()() function. Multiple optimizations are logically OR'd together.
Callbacks=	Causes kcmstest to call the KcsSetCallback()() function. Callbacks are registered in the log file.
XStatus=	The default expected status is KcsSuccess. If the script command is expected to complete successfully, the XStatus= keyword is not required. In cases where a script command is expected to return a non- success status, the XStatus= is followed by the corresponding expected, non-success status in hexadecimal format.

Table 3–17 presents the OPTIMIZE: command Optimization= keyword values.

TABLE 3–17 OPTIMIZE: Command Optimization= Keyword Values

Value	Optimization Type Set
None	KcsOptNone
Accuracy	KcsOptAccuracy
Speed	KcsOptSpeed
Size	KcsOptSize

SAVE:

SAVE: Command Description

SAVE: functionality corresponds to the KcsSaveProfile()() call. When this command is interpreted, theKcsSaveProfile()() function is executed and the status is reported back from the kcmstest command display to the test log.

SAVE: Command Syntax Example

```
SAVE:Reference=connected; File Name=modlhtst.pro;
```

SAVE: Keywords and Values

Table 3–18 presents the SAVE: command keywords and their descriptions.

TABLE 3–18 SAVE: Command Keywords

Keyword	Description
`Reference=`	Is the name that was assigned when the profiles were loaded or created, via the `CONNECT:` command.
`File Name=`	Is the name of the file to which the profile is saved.
`XStatus=`	The default expected status is `KcsSuccess`. If the script command is expected to complete successfully, the `XStatus=` keyword is not required. In cases where a script command is expected to return a non- success status, the `XStatus=` is followed by the corresponding expected, non-success status in hexadecimal format.

SETATTR:

SETATTR: Command Description

`SETATTR:` functionality corresponds to the `KcsSetAttribute()()` call. When this command is interpreted, the `KcsSetAttribute()()` function is executed and the status is reported back from the `kcmstest` command display to the test log.

See Chapter 5 ," for a test script example showing how to set each supported attribute.

SETATTR: Command Syntax Example

```
SETATTR:Reference=scanner; Attribute Tag=icSigCopyrightTag;
Attribute Value=SUN MICROSYSTEMS 1996;
```

SETATTR: Keywords and Values

Table 3–19 presents the `SETATTR:` command keywords and their descriptions.

TABLE 3–19 SETATTR: Command Keywords

Keyword	Description
`Reference=`	Is the name that was assigned when the profiles were loaded, created, or connected.
`Attribute Tag=`	Is the name of the attribute that is to be manipulated by this command. For a list of all the attribute names and examples of how to set values for them, see Chapter 5 .
`Attribute Value=`	Is the value to be applied to the attribute identified in the `Attribute Tag=` keyword. If the attribute type is a string, insert the string after the command and follow it with a semi-colon. If the attribute type is an integer or a float value, enter the corresponding string value into the script and follow it with a semi-colon. If more than one value is required, separate the values with commas (,). Finally, if the attribute type is an enumerated type, see Chapter 5, "KCMS Profile Attributes," in the *KCMS Application Developer's Guide* for a description of the enumerated types. See Chapter 5 ," in this guide, for examples of setting attributes.
`XStatus=`	The default expected status is `KcsSuccess`. If the script command is expected to complete successfully, the `XStatus=` keyword is not required. In cases where a script command is expected to return a non- success status, the `XStatus=` is followed by the corresponding expected, non-success status in hexadecimal format.
`Count=`	Is the count of data values (of type `AttributeType=`, where appropriate) found after the `Attribute Value=` keyword when creating a new attribute for an ICC profile.

UPDATE:

UPDATE: Command Description

`UPDATE:` functionality corresponds to the `KcsUpdateProfile()()` call. When this command is interpreted the `KcsUpdateProfile()()` function is executed, and the status is reported back from the `kcmstest` command display to the test log.

UPDATE: Command Syntax Example

```
UPDATE:Reference=umax; Profile Type=Scan; Operation=Both;
CharInDataFile=umax_char.aim; CharOutDataFile=umax_char.mea;
CalInDataFile=umax_cal.aim; CalOutDataFile=umax_cal.meas;
```

UPDATE: Keywords and Values

Table 3–20 presents the UPDATE: command keywords and their descriptions.

TABLE 3–20 UPDATE: Command Keywords

Keyword	Description
Reference=	Is the name that was assigned when the profiles were loaded or created, with the CONNECT: or CREATE: command.
Profile Type=	Is the type of profile that is being updated. The acceptable types are Print Scan Mon Effect Currently only scanner and monitor profiles can be updated.
Operation=	Is the type of operation that is being attempted during the update. The acceptable operations are Characterization Calibration Both
CalInDataFile=	Is the data file name of the calibration input data. It is assumed that the file is in the kcmstest/data directory.
CalOutDataFile=	Is the data file name of the calibration output data. It is assumed that the file is in the kcmstest/data directory.
CharInDataFile=	Is the data file name of the characterization input data. It is assumed that the file is in the kcmstest/data directory. This value can be NULL.

TABLE 3–20 UPDATE: Command Keywords *(continued)*

Keyword	Description
`CharOutDataFile=`	Is the data file name of the characterization output data. It is assumed that the file is in the `kcmstest/data` directory. This value can be `NULL`.
`XStatus=`	The default expected status is `KcsSuccess`. If the script command is expected to complete successfully, the `XStatus=` keyword is not required. In cases where a script command is expected to return a non- success status, the `XStatus=` is followed by the corresponding expected, non-success status in hexadecimal format.

CHAPTER 4

KCMS Test Script Descriptions

In This Chapter

This chapter describes each test script in the KCMS test suite. The chapter groups the test scripts into the categories listed in Table 4–1 and presents them in the order shown (that is, loading profiles is presented first, connecting profiles second, and so forth).

Test Script Categories

Table 4–1 shows the KCMS API function name and the corresponding script command name that is used by the `kcmstest` utility. When describing a function being performed, this chapter uses the test script command name.

TABLE 4–1 Test Script Categories

Category	KCMS "C" API Function	Script Command
Loading profiles	`KcsLoadProfile()()`	`LOAD:`
Connecting profiles	`KcsConnectProfile()()`	`CONNECT:`
Evaluating profiles	`KcsEvaluateProfile()()`	EVAL:

TABLE 4–1 Test Script Categories *(continued)*

Category	KCMS "C" API Function	Script Command
Optimizing profiles	`KcsOptimizeProfile()()`	`OPTIMIZE:`
Modifying load hints	`KcsModifyLoadHints()()`	See "Cross-Category API Functions And Script Commands" on page 40 .
Saving profiles	`KcsSaveProfile()()`	See "Cross-Category API Functions And Script Commands" on page 40 .
Getting attributes	`KcsGetAttribute()()`	`GETATTR:`
Setting attributes	`KcsSetAttribute()()`	`SETATTR:`
Updating profiles	`KcsUpdateProfile()()`	`UPDATE:`
Freeing profiles	`KcsFreeProfile()()`	See "Cross-Category API Functions And Script Commands" on page 40 .
Enhancements	No particular function; tests new features and bug fixes	

Cross-Category API Functions And Script Commands

The `KcsAvailable()()`, `KcsCreateProfile()()`, `KcsFreeProfile()()`, `KcsModifyLoadHints()()`, `KcsSaveProfile()()`, and `KcsSetCallback()()` functions in the KCMS framework API are not addressed directly as a testing category in a single script. Each of these functions is exercised in the course of performing normal testing.

- `KcsCreateProfile()` (`CREATE:` command) is called to generate an empty profile that can be used by subsequent script commands such as `GETATTR:`, `SETATTR:`, and `UPDATE:`.
- `KcsFreeProfile()()` (`FREE:` command) is called in each test script where a profile is loaded, connected, or created. This is the expectation of the KCMS framework.

- `KcsModifyLoadHints()()`(`MODIFYLH:` command) typically is called to load the rest of a profile previously loaded for attributes only. It is called in the `IC_lhints.scr` script.
- `KcsSaveProfile()()` (`SAVE:` command) is performed and tested in several of the script categories listed in Table 4–2 .
- `KcsSetCallback()()` is called in the `EVAL:`, `UPDATE:`, and `OPTIMIZE:` script commands whose operation is expected to take an extended period of time.

In addition, the `LOG:` command is not associated with a particular KCMS API function. Instead it serves to show comment data in the test scripts.

TABLE 4–2 Testing the `SAVE:` Command

Category	Script Name
Connecting Profiles	`IC_conerr.scr`
Getting and Setting Attributes	`IC_attr1.scr`
Updating Profiles	`IC_update1.scr`, `IC_update2.scr`
Enhancements	`IC_gray.scr`, `IC_pacbug.scr`, `IC_sun_update.scr`, `IC_updatewin.scr`, `IC_xprofilesav.scr`, `IC_xprofilesavremote.scr`, `IC_xprofilesavroot.scr`

For More Information on API Functions

This chapter summarizes the testing of the KCMS API functions. Operation of a function is described only where it is necessary to describe the associated testing. For a detailed function descriptions, see the KCMS SDK manual *KCMS Application Developer's Guide.*

Note - Many of the images resulting from `EVAL:` calls are not saved. This is only to limit the amount of disk space used by the test suite. See "`EVAL:` Keywords and Values" on page 21 (`ImageOut=` keyword) for details on how to save the resulting image.

Loading Profiles

Load All Now

Script Name

`IC_lana.scr`

Concept

`IC_lana.scr` loads an arbitrarily large number of profiles and verifies that profiles can be connected and an image evaluated.

Description

This script demonstrates that the KCMS framework can load and maintain several profiles in memory at the same time. The load hint specified in each of the load commands is `AllNow`. (See Table 3–11 for the `LOAD:` command `LoadHint=` keyword values and the corresponding load hints.) The type of profiles loaded varies (monitor, printer, scanner, color space). The script first loads all the profiles, after which it performs some simple operations such as `CONNECT:` and `EVAL:` to demonstrate that the framework can operate under these conditions. Then all the profiles are freed from memory with the `FREE:` command.

Verification

Each command is expected to return a successful status. Examine the test images output by the `EVAL:` command. Do not make color quality evaluations unless you have the appropriate devices to do so.

Load Many

Script Name

`IC_lmany.scr`

Concept

`IC_lmany.scr` performs many loads but with few profiles in memory at any one time. It confirms that subsequent profile-related operations can be performed successfully.

Description

This script demonstrates that the KCMS framework can load many profiles in succession while the framework continues to operate without error. The type of profiles loaded vary (monitor, printer, scanner, color space), and the profiles are loaded with the `AllNow` load hint. (See Table 3–11 for the `LOAD:` command `LoadHint=` keyword values and the corresponding load hints.) In general, the script simply loads the profiles and immediately frees them. After loading and freeing 100 profiles, it loads more profiles, connects them, and evaluates some images. Then it frees the profiles with the FREE: command.

Verification

Each command is expected to return a successful status. Examine the test images output by the `EVAL:` command. Do not make color quality evaluations unless you have the appropriate devices to do so.

Load Hints Test

Script Name

`IC_lhints.scr`

Concept

`IC_lhints.scr` loads profiles, varying the load hints applied. It performs subsequent operations, verifying that all parts of a profile required for an operation get automatically loaded if they were not specified in the load hints.

Description

This script demonstrates that the KCMS framework can perform a `LOAD:` command with a variety of load hints applied. The script performs additional functions to verify the automatic loading of profiles. For example, when a profile is loaded specifying attributes only, it is expected that a `CONNECT:` command can complete

successfully without having to manually load the remainder of the profile. The type of profiles loaded vary (monitor, printer, scanner, color space). The load hints are broken down into categories similar to the ones in the *KCMS Application Developer's Guide* (that is, what, how, when, and where to load and unload a profile). The script mixes these various load hint categories and loads several profiles. After these operations are completed all the profiles are freed from memory.

Note - The operation load hints forward, reverse, simulate, and gamut have no effect in the `LOAD:` command. Unless attributes only is specified, all available transformations are loaded. Even if attributes only is specified, `KcsConnectProfiles()()`automatically loads all the transformations. This is provided, however, for CMM developers who provide these capabilities in their CMMs.

The script performs the following operation sequence:

1. It loads scanner and monitor profiles specifying various load hints. It attempts to connect the profiles.
2. It loads scanner and printer profiles, specifying load when needed, unload when needed, and the image content hint settings. It attempts to connect the profiles, specifying the forward transformation. It verifies that the profile can be connected.
3. It loads monitor and printer profiles, specifying the graphics content hint. It connects the profiles, specifying forward and graphics. It evaluates an image using this complete transformation and verifies success.
4. It loads scanner and monitor profiles, specifying the content unknown hint. It connects the profiles, specifying forward and unknown. It evaluates using this complete transformation and verifies success.
5. It loads a PhotoCD profile, specifying attributes only. It gets all the profile attributes.
6. It loads scanner, monitor, and printer profiles, specifying the content unknown hint. It creates complete transformations for each of the following paths and evaluates using these transformations:
   a. scanner ->printer (forward)
   b. printer -> monitor reverse)
   c. monitor->printer->monitor (simulate)

To save disk space, it does not output the images resulting from the `EVAL:` commands. This can be changed if you have enough disk space on your system. See "`EVAL:` Keywords and Values" on page 21 (`ImageOut=` keyword) for details on how to save the resulting image.

Note - To date, profiles with an image, graphics, or a content unknown hint are not available. Since image, graphics, and colorimetric content hints execute the same code anyway, these tests should complete successfully. This is provided, however, for CMM developers who provide this functionality in their CMMs.

Verification

All commands are expected to return successfully.

Connecting Profiles

Connect Profiles

Script Name

`IC_connect.scr`

Concept

`IC_connect.scr` connects various types of device profiles into complete profiles. It evaluates using the complete profiles.

Description

This script demonstrates that the KCMS framework can connect a variety of profiles in a variety of ways. It loads profiles of the following types: scanner, monitor, and printer. After the profiles are loaded, the script makes a variety of connections with the `CONNECT:` command.

Initially the script works with scanner and monitor profiles, creating complete profiles, one for each of the following transformation types: forward, reverse, and gamut. It evaluates using the profiles containing the forward transformations.

Next it loads monitor and printer profiles and creates complete profiles including forward, reverse, and simulate transformations. It creates additional profiles that not only contain these transformations but have image content hints specified as well. Then it evaluates using the forward and simulate profiles. To save disk space, outputs are not saved.

Finally the script works with scanner and printer profiles.

At various points, the script evaluates the connected profiles to verify that the new profile can be used to process image data. Additionally, it varies the content of the connected profile (image, content unknown, and so forth).

Verification

Each command is expected to return successfully.

Connect Many Profiles

Script Name

`IC_conmany.scr`

Concept

`IC_conmany.scr` performs many `CONNECT:` commands. It confirms that subsequent framework operations can be performed successfully.

Description

This script demonstrates that the KCMS framework can connect many profiles in succession while the framework continues to operate without error. The script loads a variety of profiles (monitor, printer, scanner, Photo CD), and connects them with a variety of operation and content hints. On the profiles loaded, it performs 20 connect calls. With the 20 connected profiles in memory, it evaluates the complete profiles. It does not save the color-managed images. After the operations are completed, it frees the profiles from memory.

Verification

Each command is expected to return successfully.

Connect Error

Script Name

`IC_conerr.scr`

Concept

`IC_conerr.scr` attempts to connect various types of device profiles into complete profiles, testing the error handling capabilities of the `CONNECT:` command.

Description

This script demonstrates that the KCMS framework can perform the `CONNECT:` command under a variety of error conditions and return the correct status. The script performs the `CONNECT:` command with the following error conditions:

- It attempts to connect a profile that only has its attributes loaded.
- It attempts to create a simulate profile with only two profiles provided.

Verification

Commands that are expected to fail will have the expected failure status provided as part of the `CONNECT:` script command.

Evaluating Profiles

Evaluate

Script Name

`IC_eval.scr`

Concept

`IC_eval.scr` performs several `EVAL:` commands on different image types (computer-generated graphics, scanned images). The resolution of the images varies

from 72 dpi to 200 dpi. It saves the TIFF file outputs of the evaluate tests to allow for subjective evaluation of color quality.

Note - Subjective evaluation of the images requires the following devices: Apple 13" monitor, Kodak XL 7720 printer, Kodak ColorEdge 1550 copier-printer. Images used in this test are scanned on a Microtek 600ZS scanner.

Description

This script demonstrates that the KCMS framework can successfully perform the `EVAL:` command under varying input and output conditions. First several profiles are loaded, and the `CONNECT:` command is used to create forward, reverse, and simulate test profiles. Test images are evaluated through each of the profiles that were previously connected. After these operations are completed, all the profiles are freed from memory with the `FREE:` command.

Verification

Each command in this script is expected to return successfully. As a post-test exercise, you should generate and examine each of the images output from the `EVAL:` command as an additional verification that the evaluate tests completed successfully.

Evaluate Gamut Range

Script Name

`IC_gamut.scr`

Concept

`IC_gamut.scr` checks the color gamut of an image it evaluates.

Description

This script demonstrates that the KCMS framework can successfully perform the `EVAL:` command and check the color gamut. First the script loads two profiles and connects them to create a resulting profile. It then evaluates the image through the resulting profile, requesting that the color gamut be checked. Since not all devices can represent the same number or range of colors, gamut testing can indicate how many of the image's colors are reproducible on the output device.

Verification

The number of pixels that are out of gamut (that is, their colors are not reproducible) is printed. The output image is not saved but instead is represented by 0's and FF's. Each 0 represents an in-gamut pixel and each FF, an out-of-gamut pixel.

Evaluate Many

Script Name

`IC_evalmany.scr`

Concept

`IC_evalmany.scr` evaluates many images, processing over 100 MB of image data.

Description

This script demonstrates that the KCMS framework can successfully perform the `EVAL:` command repetitively. First the script loads several profiles, and the `CONNECT:` command is called to create forward, reverse, and simulate profiles. In all, this test performs 25 `EVAL:` calls, constituting the processing of over 100 MB of image data. The script does not save the resulting evaluated image.

For details on how to save the image, see "`EVAL:` Keywords and Values" on page 21 (`ImageOut=` keyword). Be sure you have enough disk space to do so.

Verification

Each command is expected to return successfully.

Evaluate Layout

Script Name

`IC_layouts.scr`

Concept

`IC_layouts.scr` evaluates images with the organization of the image data varied. The image organizations tested are: RGB row interleaved, RGB interleaved, and RGB planar. `kcmstest` is responsible for organizing the image data in the specified format. The script processes graphic images and saves the output for later evaluation. In all cases, the output images should be saved in RGB interleaved image organization so they can be examined after the test is completed.

Description

This script demonstrates that the KCMS framework can successfully perform the `EVAL:` command under varying input and output conditions. First the script loads several profiles and uses the `CONNECT:` command is to create forward, simulate test, and reverse profiles. Test images are passed through these profiles, exercising various image organizations specified for the input image. After these operations are completed, all the profiles are freed from memory.

Verification

Each command is expected to return successfully. As a post-test exercise, you can examine each of the images output from the `EVAL:` command as additional verification that the evaluate layout tests completed successfully.

Evaluate Error

Script Name

`IC_evalerr.scr`

Concept

`IC_evalerr.scr` attempts to create test conditions that cause the `EVAL:` command to return various errors.

Description

This script demonstrates that the KCMS framework can perform a `CONNECT:` command under a variety of error conditions and return the appropriate error status. The script performs the `CONNECT:` command with the following error conditions:

- It attempts to evaluate using a profile that is not complete.

- It attempts to evaluate an image, specifying a transformation that is not part of the complete profile provided in the `EVAL:` command.
- It attempts to evaluate an image, specifying a content hint that is not part of the complete profile provided in the `EVAL:` command.

Verification

Commands that are expected to fail will have the expected failure status provided as part of the `EVAL:` script command.

Optimizing Profiles

Speed Optimization

Script Name

`IC_optspeed.scr`

Concept

`IC_optspeed.scr` evaluates test images using complete profiles it creates with forward, reverse, and simulate transformations. It optimizes the profiles for speed and again evaluates the images.

Description

This script demonstrates that the KCMS framework can create complete profiles with forward, reverse, and simulate transformations and can optimize them for speed. The script loads a scanner, monitor, and printer profile and creates three connected profiles with the forward, reverse, and simulate operations. It evaluates each connected profile and records the time required to perform each evaluation in the log file. Then it optimizes the connected profiles for speed and repeats evaluations. The time required to perform each evaluation is again recorded to the log file. The script does not save the color-managed images.

Verification

Each of the commands performed in this test is expected to complete successfully. The optimized profiles are expected to reduce the time required to evaluate the image.

Size Optimization

Script Name

`IC_optsize.scr`

Concept

`IC_optsize.scr` creates complete profiles that have forward, reverse, and simulate transformations and gets the profile sizes. It optimizes the complete profiles for size. Then it gets the new profile sizes. The profiles are written to the log file.

Description

This test script demonstrates that the KCMS framework can create complete profiles with forward, reverse, and simulate transformations and can optimize them for size. It uses scanner, color space, and printer profiles to test the `OPTIMIZE:` command for size. It creates complete profiles, specifying the forward, reverse, and simulate transformations. Using the `GETATTR:` command, it gets the size of the complete profiles. Once this is done, it uses the `OPTIMIZE:` command to optimize an image for size. Again the script gets the size of each of the completed profiles.

Verification

Each of the commands performed is expected to complete successfully. The optimized profile sizes are expected to be reduced from the original sizes.

Getting and Setting Attributes

Get/Set Attribute

Script Name

`IC_attr1.scr`

Concept

`IC_attr1.scr` sets attributes for a variety of device profiles, which it saves and frees from memory. Then it reloads the profiles and verifies that the attributes were correctly set. It varies the attributes that it sets and gets.

Description

This script demonstrates that the KCMS framework can perform `SETATTR:` commands for scanner, printer, and monitor profiles and verify that the attributes were correctly set. Prior to operating on each profile type, the script uses the `GETATTR:` command to retrieve all the attributes for that profile. After setting the attributes with the `SETATTR:` command, the script saves and then frees each profile from memory. Then it reloads the profile and performs a `GETATTR:` command on the attributes previously set. You should examine the log file to verify that:

- Only the profile attributes modified with the previously executed `SETATTR:` commands have been modified.
- The modified attributes reflect the values defined in the `SETATTR:`command.

Verification

All the commands performed in this test are expected to complete successfully. Additionally, the profile attributes are expected to be modified to the values specified in the `SETATTR:` commands.

Attribute Test 2

Script Name

`IC_attr2.scr`

Concept

`IC_attr2.scr` creates a new profile and sets a variety of attributes. It gets the attributes and verifies that they have been properly set. It saves the profile, frees it, reloads the saved profile, and again gets the attributes.

Description

This script demonstrates that the KCMS framework can set the attributes of a profile it creates. After it creates the profile, it uses the `SETATTR:` command to set the

attributes. Then it uses the GETATTR: command to get all the attributes it set. The script saves the profile and frees it from memory. Then it reloads the profile and again gets the attributes it set.

Verification

All the commands performed in this test are expected to be successfully completed. You should examine the log file to verify that:

- Only the profile attributes modified via the previously executed SETATTR: commands have been modified.
- The modified attributes reflect the values defined in the SETATTR: command.

Lookup Tables

Script Name

IC_lut.scr

Concept

IC_lut.scr uses data files in the data directory to set a lookup table (LUT) structure in the profile and to get the LUT from the profile.

Description

This script demonstrates that the KCMS framework can support both 8-bit and 16-bit LUTs. Not all profiles use the LUT technology within the profile, so not all profiles will have LUTs. See the *KCMS Application Developer's Guide* for more information on the types of LUTs.

The SETATTR: command takes the name of the data file in the data directory containing the LUT structure of values.

The GETATTR: command prints out the LUTs. The data can be very large—75,000 values. Once the LUTs are set, the profile must be saved before they are actually written into and accessed by the GETATTR: command.

Verification

Examine the log to verify that the LUT values printed out match the LUT values in the data files.

Updating Profiles

Update Scanner Profile

Script Name

`IC_update1.scr`

Concept

`IC_update1.scr` creates and updates a scanner profile with HP Scanjet calibration and characterization data. It connects the scanner profile with a monitor profile to create a complete profile. The complete profile is then used to evaluate an image. The resulting image is saved for post-test subjective evaluation.

Description

This script demonstrates that the KCMS framework can create and update a scanner profile with calibration and characterization data. The data used will not necessarily match your scanner.

The test script verifies that, after the profile is updated, it can be connected to a monitor profile and the resulting profile used to evaluate images.

The script creates a scanner profile and sets several attributes after which it performs the `UPDATE:` command. Then it saves the updated profile and connects it to a monitor profile. It evaluates the test image using this complete profile. Then it frees the profiles from memory.

Verification

All the commands listed in this test are expected to complete successfully. You need to subjectively evaluate the image resulting from the `EVAL:` command.

Update Monitor Profile

Script Name

`IC_update2.scr`

Concept

`IC_update2.scr` creates and updates a monitor profile with calibration data. Monitor profiles in this test contain no characterization data; however the `SETATTR:` command must set the monitor white point and the CIEXYZ chromaticity for the red, green, and blue phosphors.

The monitor white point and chromaticity for the red, green, and blue phosphors of ICC profiles are defined in CIEXYZ color space. After the profile is created, it is connected with a printer profile to create a complete profile.

Note - The scanner profile will have been previously verified. The complete profile is then used to evaluate the image. The resulting image is saved for post-test subjective evaluation.

Description

This script demonstrates that the KCMS framework can create and update a monitor profile with calibration data. It uses the Sony 16” monitor profile distributed with the KCMS product and updates it with the appropriate monitor calibration data. The resulting data may not match your system characteristics.

The test script examines monitor profile updating. It loads the monitor profile and performs the appropriate `SETATTR:` commands. It then performs the `UPDATE:` command. It saves the updated profile and frees it from memory. It reloads the profile and connects it to a printer profile. Using this complete profile, it evaluates the test image.

Verification

All the commands in this test are expected to completed successfully. You need to subjectively evaluate the image resulting from the `EVAL:` command.

Enhancement Tests

The scripts listed below are described alphabetically by script name. These scripts test value-added features and bug fixes to the KCMS framework.

- `IC_evalplus.scr`
- `IC_gray.scr`
- `IC_loadsol.scr`
- `IC_pacbug.scr`

- IC_sun_update.scr
- IC_updatewin.scr
- IC_xdisplay.scr
- IC_xprofile.scr
- IC_xprofilehost.scr
- IC_xprofilesav.scr
- IC_xprofilesavremote.scr
- IC_xprofilesavroot.scr
- IC_xwindow.scr
- IC_xwindowerr.scr

IC_evalplus.scr

Concept

IC_evalplus.scr connects configured X Window System visual profiles to scanner and printer profiles and evaluates the profiles in the same manner as generic profiles.

Description

This script is similar to IC_eval.scr (see “Evaluate” on page 47)with one exception: the monitor profile used is a configured/calibrated X Window System visual profile for the current frame buffer. This test should succeed if the system has been previously configured using the kcms_configure(1) or kcms_calibrate(1) command. After evaluation, the script frees the profiles from memory.

Verification

All the commands listed in this script are expected to complete successfully.

IC_gray.scr

Concept

IC_gray.scr creates a gray profile and sets attributes.

Description

This script demonstrates that the KCMS framework can create a gray device color profile and set several attributes. It performs the following sequence of events first on a display profile and then on an input profile.

It uses the `CREATE:` command to create the new profile and sets several attributes with the `SETATTR:` command. Then it uses the `GETATTR:` command to get the attributes to verify that they are properly set. It saves the profile and frees it from memory. Then it reloads the profile and again verifies the attributes.

Verification

All the commands are expected to complete successfully.

IC_loadsol.scr

Concept

`IC_loadsol.scr` loads and frees 100 59

Solaris file-type profiles without memory problems.

Description

This script demonstrates that the KCMS framework successfully can load and free 100 Solaris™ file-type profiles. The script actually verifies a previous fix of an error that caused file descriptors to overflow because of improper file closings in the library.

Verification

All the commands in this script are expected to complete successfully.

IC_pacbug.scr

Concept

`IC_pacbug.scr` verifies a bug in the `CONNECT:` command.

Description

This script tests scanner and monitor profile connects. It tests a previous bug in the system.

Verification

All the commands complete as expected.

IC_sun_update.scr

Concept

`IC_sun_update.scr` verifies a bug in the `UPDATE:` command.

Description

This script demonstrates that the KCMS framework can update a monitor profile several times. It loads, updates, and saves a monitor profile to a different name three times and frees the updated profiles. Then the script reloads each of the saved monitor profiles and a scanner profile. It connects each monitor profile to the scanner profile, specifying the forward transformation operation and evaluates the results. Finally, the script frees all the profiles from memory.

Verification

All the commands are expected to complete successfully.

IC_updatewin.scr

Concept

`IC_updatewin.scr` updates a profile several times in a row.

Note - This script must be run as root.

Description

This script demonstrates that the KCMS framework can update an X Window System profile several times–saving and freeing the profile each time.

Verification

All the commands are expected to complete successfully.

IC_xdisplay.scr

Concept

`IC_xdisplay.scr` recognizes a display number when accessing a remote host.

Description

This script demonstrates that the KCMS framework can recognize a display number when accessing a remote host. It requests a Solaris file profile from display 0.0 of a remote host that has a single display. It loads a local X Window System profile and sets an attribute to verify that the host is reset properly. Then it frees the profiles from memory.

Note - The `kcms_server(1)` daemon must be running on the remote most. If it is not running, type **`kcms_server`** in a command shell as root on the remote host.

Verification

All the commands in this test are expected to complete successfully provided the `kcms_server(1)` daemon is running.

IC_xprofile.scr

Concept

`IC_xprofile.scr` tests ways of finding profiles.

Description

`IC_xprofile.scr` sets the environment variable `KCMS_PROFILES` to a directory containing a profile called `junk1.pro`, which should be a copy of an existing profile copied to this directory prior to running the test. It essentially tests the ability of the library to use `KCMS_PROFILES` to find profiles.

Verification

All the commands in this test are expected to complete successfully.

IC_xprofilehost.scr

Concept

`IC_xprofilehost.scr` tests local and remote hosts.

Description

This script finds a profile remotely and sets the KCMS host to a remote server. It tests the local host using the Internet name instead of the keyword `local` and sets the KCMS host to the local machine.

Verification

All the commands in this test are expected to complete successfully.

IC_xprofilesav.scr

Concept

`IC_xprofilesav.scr` saves an X Window System profile.

Description

This script attempts to save X Window System profiles in `/etc/openwin/devdata` without being root.

Verification

The test should fail with `Xstatus 4011` (`KCS_IO_WRITE`), because it does not have write permissions.

IC_xprofilesavremote.scr

Concept

`IC_xprofilesavremote.scr` tests writing to a remote host.

Description

This script sets the `DISPLAY` environment variable to a KCMS remote host. Then it attempts to save an X Window System profile remotely.

Verification

The test should fail with `XStatus 4302(KCS_X11_PROFILE_RO)`, because it does not have write permissions.

IC_xprofilesavroot.scr

Concept

`IC_xprofilesavroot.scr` saves an X Window System profile.

Note - This script must be run as root.

Description

This script saves an X Window System profile in `/etc/openwin/devdata/profiles`.

Verification

All the commands in this test are expected to complete successfully.

IC_xwindow.scr

Concept

`IC_xwindow.scr`:

- Automatically accesses a profile that has been previously configured for the current frame buffer using the `kcms_configure(1)` or `kcms_calibrate(1)` commands
- Accesses a Solaris profile across the network

Description

This script demonstrates that the KCMS framework can access profiles locally and across the network. First it loads the default X Window System profile from the local host. Then it loads a Solaris profile from another host machine. Finally it frees the profiles from memory.

Note - The `kcms_server(1)` daemon must be running on the remote most. If it is not running, type **`kcms_server`** in a command shell as root on the remote host.

Verification

All the commands are expected to complete successfully, provided the `kcms_server(1)` daemon is running on the current machine.

IC_xwindowerr.scr

Concept

`IC_xwindowerr.scr` captures errors and reports `KcsStatus` class extensions.

Description

This script tests Solaris file error cases. An invalid host name is requested in three `LOAD:` commands, and invalid profiles are requested from the valid local host in three other `LOAD:` commands.

Verification

All the `LOAD:` commands should fail because of an attempt to access an invalid host name or profile name.

CHAPTER 5

Setting Attributes

In This Chapter

This chapter provides a script example showing how to use the SETATTR: command to set attributes. The chapter includes an example for each supported attribute. Examples are presented alphabetically by attribute name. In most cases, the values exactly match the fields in the icHeader structure. Those that don't are indicated.

icSigHeaderTag

```
Attribute
Tag=icSigHeaderTag; Attribute Value=KCMS,2,icSigOutputClass,icSigRgbData,

       icSigLabData,95,7,27,17,30,15,acsp,icSigSolaris,

1,prnt,test,0,0,0,0.964188,1.0,0.82489;
```

Attribute value= are values that exactly match the fields in the icHeader structure. See the icHeader structure in icc.h.

icSigAToB0Tag

You set and get each of the following attributes in the same manner:

- icSigAToB0Tag
- icSigAToB1Tag
- icSigAToB2Tag

```
Attribute
Tag=attribute_name Attribute

Value=file_name
```

`Attribute Tag=` is `icSigAToB0Tag`, `icSigAToB1Tag`, or `icSigAToB2Tag`.

`Attribute Value=` is the name of a file containing data in the structure of an `ic_lut8Type` or `ic_lut16Type`. See the the `icc.h` header file for a description of each of these structures.

icSigBlueColorantTag

```
Attribute
Tag=icSigBlueColorantTag; Attribute

Value=29.41,12.37,151.21;
```

`Attribute Value=` are floating point X, Y, and Z values.

icSigBlueTRCTag

```
Attribute
Tag=icSigBlueTRCTag; Count=4; Attribute

Value=0,30000,45000,65535;
```

`Count=` is the number of values to be supplied.

`Attribute Value=` are 2-byte values.

icSigBToA0Tag

You set and get each of the following attributes in the same manner:

- `icSigBToA0Tag`
- `icSigBToA1Tag`
- `icSigBToA2Tag`

```
Attribute
Tag=attribute_name Attribute

Value=file_name
```

`Attribute Tag=` is `icSigBToA0Tag`, `icSigBToA1Tag`, or `icSigBToA2Tag`.

`Attribute Value=` is the name of a file containing data in the structure of an `ic_lut8Type` or `ic_lut16Type`. See the the `icc.h` header file for a description of each of these structures.

icSigCalibrationDateTimeTag

```
Attribute
Tag=icSigCalibrationDateTimeTag; Attribute

Value=1995,6,27,16,34,0;
```

`Attribute Value=` are the following values:

1. Year
2. Month
3. Day
4. Hour
5. Minutes
6. Seconds

icSigCharTargetTag

```
Attribute
Tag=icSigCharTargetTag; Attribute

Value=IT8.7/2;
```

`Attribute Value=` is the ASCII string.

icSigCopyrightTag

```
Attribute
Tag=icSigCopyrightTag; Attribute Value=No Copy

right;
```

Attribute Value= is the ASCII string.

icSigDeviceMfgDescTag

```
Attribute
Tag=icSigDeviceMfgDescTag; Count=50; Attribute Value=55,56,QA

Test;
```

Count= is the number of characters in the ASCII string description. It must be greater than or equal to the actual number of characters plus a terminating NULL byte.

Attribute Value= are Unicode description length (optional), Scriptcode description length (optional), and ASCII string description.

icSigDeviceModeDescTag

```
Attribute
Tag=icSigDeviceModelDescTag; Count=50; Attribute Value=55,56,All icc

attributes;
```

Count= is the number of characters in the ASCII string description. It must be greater than or equal to the actual number of characters plus a terminating NULL byte.

Attribute Value= are Unicode description length (optional), Scriptcode description length (optional), and ASCII string description.

icSigGamutTag

```
Attribute
Tag= icSigGamutTag Attribute

Value=file_name
```

Attribute Value= is the name of a file containing data in the structure of an ic_lut8Type or ic_lut16Type. See the the icc.h header file for a description of each of these structures.

icSigGrayTRCTag

```
Attribute
Tag=icSigGrayTRCTag; Count=9; Attribute

Value=0,8191,16383,24575,32767,40959,49151,57343,65535;
```

`Count=` is the number of values to be supplied.

`Attribute Value=` are 2-byte values.

icSigGreenColorantTag

```
Attribute
Tag=icSigGreenColorantTag; Attribute

Value=46.40,100.0,22.20;
```

`Attribute Value=` are floating point X, Y, and Z values.

icSigGreenTRCTag

```
Attribute
Tag=icSigGreenTRCTag; Count=4; Attribute

Value=0,21512,43024,65535;
```

`Count=` is the number of values to be supplied.

`Attribute Value=` are 2-byte values.

icSigLuminanceTag

```
Attribute
Tag=icSigLuminanceTag; Attribute

Value=38.668,40.0,32.996;
```

`Attribute Value=` are floating point X, Y, and Z values.

icSigMeasurementTag

```
Attribute
Tag=icSigMeasurementTag; Attribute Value=icStdObs1931TwoDegrees,1.0,1.0,1.0,

icGeometry045or450,icFlare0,icIlluminantD50;
```

`Attribute Value=` are values that exactly match fields in the `icMeasurement` structure. See the `icMeasurement` structure in `icc.h`.

icSigMediaBlackPointTag

```
Attribute
Tag=icSigMediaBlackPointTag; Attribute

Value=0.056,0.12,0.003;
```

`Attribute Value=` are floating point X, Y, and Z values.

icSigMediaWhitePointTag

```
Attribute
Tag=icSigMediaWhitePointTag; Attribute

Value=0.964188,1.0,0.82489;
```

`Attribute Value=` are floating point X, Y, and Z values.

isSigNamedColor2Tag

```
Attribute
Tag=icSigNamedColor2Tag; Count=2; Attribute Value=3, 135, light, ish, Green,

100, 20, 20, 120, 83, 75, Red, 20, 100, 20, 75, 120,

83;
```

`Count=` is the number of colors.

`Attribute Value=` are the following values:

1. Number of channels associated with this profile's output color space
2. Vender-supplied flag

3. Prefix
4. Suffix
5. Count*(Color Name, PCS Coords(3), DevCoord*(Num of Channels))

For details, see icSigNamedColor2Tag in icc.h.

icSigPreview0Tag

You set and get each of the following attributes in the same manner:

- icSigPreview0Tag
- icSigPreview1Tag
- icSigPreview2Tag

```
Attribute
Tag=attribute_name Attribute

Value=file_name
```

Attribute Tag= is icSigPreview0Tag, icSigPreview1Tag, or icSigPreview2Tag.

Attribute Value= is the name of a file containing data in the structure of an ic_lut8Type or ic_lut16Type. See the the icc.h header file for a description of each of these structures.

icSigProfileDescriptionTag

```
Attribute
Tag=icSigProfileDescriptionTag; Count=50; Attribute Value=55,56,This is a

profile description;
```

Count= is the number of characters in the ASCII string description. It must be greater than or equal to the actual number of characters plus a terminating NULL byte.

Attribute Value= are Unicode description length (optional), Scriptcode description length (optional), and ASCII string description.

icSigProfileSequenceTag

Note - This attribute is read only via the GETATTR: command and can't be modified by the SETATTR: command.

---

icSigPs2CRD0Tag

```
Attribute
Tag=icSigPs2CRD0Tag; Count=30; Attribute Value=0, This is the Ps2CRD0

tag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are data type (0 = ASCII, 1 = binary) and ASCII string. (The script test only supports ASCII.)

icSigPs2CRD1Tag

```
Attribute
Tag=icSigPs2CRD1Tag; Count=30; Attribute Value=0, This is the Ps2CRD1

tag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are data type (0 = ASCII, 1 = binary) and ASCII string. (The script test only supports ASCII.)

icSigPs2CRD2Tag

```
Attribute
Tag=icSigPs2CRD2Tag; Count=30; Attribute Value=0, This is the Ps2CRD2

tag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are data type (0 = ASCII, 1 = binary) and ASCII string. (The script test only supports ASCII.)

icSigPs2CRD3Tag

```
Attribute
Tag=icSigPs2CRD3Tag; Count=30; Attribute Value=0, This is the Ps2CRD3

tag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are data type (0 = ASCII, 1 = binary) and ASCII string. (The script test only supports ASCII.)

icSigPs2CSATag

```
Attribute
Tag=icSigPs2CSATag; Count=30; Attribute Value=0, This is the Ps2CSA

tag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are data type (0 = ASCII, 1 = binary) and ASCII string. (The script test only supports ASCII.)

icSigPs2RenderingIntentTag

```
Attribute
Tag=icSigPs2RenderingIntentTag; Count=40; Attribute Value=0, This is the

Ps2RenderingIntent tag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are data type (0 = ASCII, 1 = binary) and ASCII string. (The script test only supports ASCII.)

icSigRedColorantTag

```
Attribute
Tag=icSigRedColorantTag; Attribute

Value=99.05,54.26,4.69;
```

`Attribute Value=` are floating point X, Y, and Z values.

icSigRedTRCTag

```
Attribute
Tag=icSigRedTRCTag; Count=4; Attribute

Value=0,20000,40000,65535;
```

`Count=` is the number of values to be supplied.

`Attribute Value=` are 2-byte values.

icSigScreeningDescTag

```
Attribute
Tag=icSigScreeningDescTag; Count=32; Attribute Value=60,70,This is a screening

description;
```

`Count=` is the number of characters in the ASCII string description. It must be greater than or equal to the actual number of characters plus a terminating `NULL` byte.

`Attribute Value=` are Unicode description length (optional), Scriptcode description length (optional), and ASCII string description.

icSigScreeningTag

```
Attribute
Tag=icSigScreeningTag;Count=3; Attribute Value=0,3,10000, 20000,

icSpotShapeRound,30000,40000,

icSpotShapeRound,50000,60000,icSpotShapeRound;
```

`Attribute Value=` are the following values (repeat for the number of channels):

1. screening flag
2. number of channels
3. freq
4. screen angle
5. spot shape

For details, see `icSigScreeningTag` in `icc.h`.

icSigTechnologyTag

```
Attribute
Tag=icSigTechnologyTag; Attribute

Value=icSigCRTDisplay;
```

`Attribute Value=` is an enumerated type from the ICC header file. See the `icTechnology` structure in `icc.h`.

icSigUcrBgTag

```
Attribute
Tag=icSigUcrBgTag;Count=40; Attribute Value=2,2,100,4,300,400,500,600,End of

UcrTag;
```

`Count=` is the number of characters in the ASCII string.

`Attribute Value=` are the following values:

1. Number of values in the sine ucr curve
2. The 2-byte ucr values
3. Number of values in the bg curve
4. The 2-byte bg curve values
5. ASCII string

icSigViewingCondDescTag

```
Attribute
Tag=icSigViewingCondDescTag; Count=32; Attribute Value=60,70,This is a viewing

description;
```

`Count=` is the number of characters in the ASCII string description. It must be greater than or equal to the actual number of characters plus a terminating `NULL` byte.

`Attribute Value=` are Unicode description length (optional), Scriptcode description length (optional), and ASCII string description.

icSigViewingConditionsTag

```
Attribute
Tag=icSigViewingConditionsTag; Attribute

Value=1.0,.8976,1.198,.756,.5,.034,icIlluminantD50;
```

`Attribute Value=` are values that exactly match fields in the `icViewingCondition` structure. See the `icViewingCondition` structure in `icc.h`.

CHAPTER 6

Putting It All Together

In This Chapter

This chapter threads together all the steps involved in using the KCMS test suite with your CMM. The chapter refers you to the appropriate KCMS documentation for details.

Development Environment Requirements

The KCMS packages are automatically placed in a protected directory when you load them with the `pkgadd(3)`command. Copy the packages to a writable directory for development use.

To compile programs, you must use version 4.2 of the Sun™ Visual Workshop™ C++ compiler, which is included with Sun Visual Workshop C++ 3.0.

Creating Your CMM

The *KCMS CMM Developer's Guide* and the *KCMS CMM Reference Manual* are your primary sources of information on how to create a CMM.

Setting Up Your CMM

Guidelines for setting up your CMM are described in detail in the *KCMS CMM Developer's Guide.* To set up your CMM,

1. Name your CMM according to the guidelines in Chapter 2, "CMM: A Runtime Derivative," in the *KCMS CMM Developer's Guide.* The section entitled "Configuration Requirements" explains what you need to know to load your CMM dynamically, including how to name it and how to update the `OWconfig` file.
2. Install your CMM according to the guidelines in the same chapter and section referenced in step 1.
3. Create and name the profile(s) for your CMM according to the guidelines in Chapter 2, "CMM: A Runtime Derivative," in the *KCMS CMM Developer's Guide.* The section entitled "Profiles" describes the ICC profile format and explains how to name profiles.
4. Install the profile so the KCMS framework can find it by following the guidelines in the same chapter and section referenced in step 3. Also install those profiles you want to use in the test suite in the `kcmstest/profiles` directory (see "Installing Scripts and Profiles" on page 78 ")or create a link to them.

Creating Test Scripts

If the test scripts that are packaged with the KCMS DDK are not adequate to test specific features of your custom CMM, you may decide to edit them. If you make any changes (change the names of, add, or delete scripts from the list in `icc.ini` file or customize the contents of scripts), you may need to create an alternate initialization file or run selected scripts. For details, see Chapter 2 ," in this guide.

The test scripts you create must follow the guidelines for using KCMS functions as described in the SDK manual *KCMS Application Developer's Guide.* For example, to evaluate profiles used by your CMM, your script first must connect profiles. Prior to connecting profiles, it must create or load profiles. Keep in mind that the test suite can only test attributes it knows about. If your profiles use new attributes, the test scripts cannot test them. For examples of how to set attribute values, see Chapter 5 .

Installing Scripts and Profiles

So that the `kcmstest` command can find them, you must install all test scripts in the `kcmstest/script` directory. Install all profiles you want to use in the test suite in the `kcmstest/profiles` directory. (See "Required File Hierarchy " on page 6 for a description of the KCMS test suite directory hierarchy.) Note that this profile

installation is a *separate* installation from the one to set up your CMM (described in “Setting Up Your CMM” on page 78 .)

Note - You may choose to install links to the location of your profiles.

Testing and Inspecting Results

Follow the guidelines for running the test scripts described in Chapter 2 . If you just plan to run a few scripts, you can use the `kcms_qatest` command with command options. See “Using `kcmstest` To Run Test Scripts” on page 10 in Chapter 2. Alternately, if you plan to run a large batch of scripts, the chapter suggests that you use the automated test scripts to do so. See “Using Automated Script Files To Run Test Scripts” on page 14 in Chapter 2.

Checking Status Codes

When you have run the scripts, inpect the log file(s).

In Chapter 2, “Running KCMS Test Scripts,” Code Example 2–3 shows the log file output for the script shown in Code Example 2–2 in that same chapter. Status codes return the value `0` if a command completes successfully. Some scripts, however, expect an error to be returned. You can use the `XStatus` keyword to test for error conditions you expect to occur.

The `IC_evalerr.scr` test script, for example, creates test conditons in which the `EVAL:` command generates errors. The `EVAL:` command includes the optional keyword `XStatus` for reporting expected errors. Code example 5-1 is an excerpt from the `IC_evalerr.scr` script. The example shows two `EVAL:` commands that will generate errors because of incorrect or missing information. In each case, `XStatus` is set to the value `4024 (‘‘KCS_PROF_NO_DATA_SUPPORT_4_REQUEST”)`

See Appendix A ,” for a list of all the status strings and their values. You also can find status codes and strings in the header file `kcsstats.h`. For additional information on the meaning of status codes, see Chapter 6, “Warnings and Error Messages,” in the SDK manual *KCMS Application Developer’s Guide.*

CODE EXAMPLE 6–1 Using `XStatus` to Report Expected Errors

```
LOG:Attempt to evaluate
with a profile that does not have the correct transform;

EVAL:Reference=forward;  SourcePixLayout=RGBInterLeaved;

 DestPixLayout=RGBInterLeaved;  ImageIn=macbeth_1550.tif;  ImageOut=None;

 Operation=Reverse;  XStatus=4024; LOG:Attempt to evaluate an image with a
```

```
content not available in the profile; EVAL:Reference=simulate;
 SourcePixLayout=RGBInterLeaved;  DestPixLayout=RGBInterLeaved;
 ImageIn=macbeth_1550.tif;  ImageOut=None;  Operation=Image;
 Operation=Reverse;  XStatus=4024;
```

APPENDIX **A**

Status Codes

In This Appendix

Table A–1 lists all the KCMS “C” API status code strings and their associated values. For additional information on the status codes, see the header file `kcsstats.h` and the KCMS SDK manual *KCMS Application Developer’s Guide*.

Note - I/O errors occur normally when you do not have enough swap space to continue.

TABLE A–1 Status Code Strings and Their Values

String	Value
Successful Status	
`KCS_SUCCESS`	`0x0000`
Warning Status	
`KCS_WARNINGS_START`	`0x1000`
`KCS_OPERATION_CANCELLED`	`0x1001`
`KCS_TRUNCATED`	`0x1002`

TABLE A–1 Status Code Strings and Their Values *(continued)*

String	Value
KCS_SPEC_CMM_NOT_FOUND	0x1003
KCS_CANNOT_OPTIMIZE	0x1004
KCS_CANNOT_DEOPTIMIZE	0x1005
KCS_ATTR_LARGE_CT_SUPPLIED	0x1006
Failure Status–General	
KCS_ERRORS_START	0x4000
Failure Status–Memory	
KCS_MEM_ALLOC_ERR	0x4006
KCS_MEM_ADDRESS_ERR	0x4007
Failure Status–Operating System	
KCS_OS_ERROR	0x4008
I/O Errors	
KCS_IO_READ	0x4010
KCS_IO_WRITE	0x4011
KCS_IO_SEEK	0x4012
KCS_IO_UNKNOWN_TYPE_ERROR	0x4013
Profile	
KCS_PROF_ID_BAD	0x4020
KCS_PROF_FORMAT_BAD	0x4021
KCS_PROF_CT_EXCEEDS_PROF_LIST	0x4022

TABLE A–1 Status Code Strings and Their Values *(continued)*

String	Value
KCS_PROF_INCOMPLETE	0x4023
KCS_PROF_NO_DATA_SUPPORT_FOR_REQUEST	0x4024
KCS_PROF_REQ_ATTRS_INCOMPLETE	0x4025
Attributes	
KCS_ATTR_NAME_OUT_OF_RANGE	0x4030
KCS_ATTR_TYPE_UNKNOWN	0x4031
KCS_ATTR_LOAD_FORMAT_INCORRECT	0x4032
KCS_ATTR_LOAD_FLOAT_ERR	0x4033
KCS_ATTR_LOAD_INT_ERR	0x4034
KCS_ATTR_DATE_TIME_FORMAT	0x4035
KCS_ATTR_CT_ZERO_OR_NEG	0x4036
KCS_ATTR_READ_ONLY	0x4037
KCS_ATTR_TYPE_NOT_SIMPLE	0x4038
Connection	
KCS_CONNECT_FAILED	0x4040
KCS_CONNECT_PRECISION_UNACCEPTABLE	0x4041
KCS_CONNECT_OPT_FORCED_DATA_LOSS	0x4042
KCS_CONNECT_PROFILES_CT_ERR	0x4043
KCS_CONNECT_QUANT_MISMATCH	0x4044

TABLE A–1 Status Code Strings and Their Values *(continued)*

String	Value
KCS_CONNECT_UNIMP_OP	0x4045
KCS_NOT_AVAILABLE	0x4054
Validation	
KCS_MISMATCHED_WHITEPOINTS	0x4060
KCS_MISMATCHED_BLACKPOINTS	0x4061
KCS_MISMATCHED_COLORSPACES	0x4062
KCS_MISMATCHED_DIMENSIONS	0x4063
KCS_MISMATCHED_VERSIONS	0x4064
Layout	
KCS_LAYOUT_INVALID	0X4070
KCS_LAYOUT_UNSUPPORTED	0X4071
KCS_LAYOUT_MISMATCH	0X4072
Evaluation	
KCS_EVAL_TOO_MANY_CHANNELS	0x4080
KCS_EVAL_BUFFER_OVERFLOW	0x4081
KCS_EVAL_ONLY_ONE_OP_ALLOWED	0x4082
Characterization / Calibration	
KCS_CC_UPDATE_NEEDS_MORE_DATA	0x4090
KCS_CC_UPDATE_INVALID_DATA	0x4091

TABLE A–1 Status Code Strings and Their Values *(continued)*

String	Value
KCS_CC_INCORRECT_COLOR_SPACE	0x4092
KCS_CC_NUM_COMPS_OUT_OF_RANGE	0x4093
KCS_CC_TOO_FEW_MEASUREMENTS	0x4094
KCS_CC_TABLE_DATA_BAD	0x4095
KCS_CC_INCORRECT_DEV_TYPE	0x4096
KCS_CC_INCORRECT_ATTR_CLASS	0x4097
KCS_CC_CANNOT_CALL_DEV_TYPE	0x4098
KCS_CC_CANNOT_CHAR_DEV_TYPE	0x4099
KCS_CC_INPUT_NOT_RAMP	0x409A
Color Management Module	
KCS_CMM_UNKNOWN_TECHNOLOGY	0x4100
KCS_COMP_MGR_FAILURE	0x4101
KCS_CMM_UNKNOWN_RUNTIME_TYPE	0x4102
KCS_CMM_UNSUPPORTED_OP	0x4103
KCS_CMM_RTLOAD_FAILED	0x4104
KCS_CMM_MAJOR_VERSION_MISMATCH	0x4105
KCS_MINOR_VERSION_MISMATCH	0x4106
Unimplemented Features	
KCS_UNIMP_NESTED_CONNECTIONS	0x4110

TABLE A–1 Status Code Strings and Their Values *(continued)*

String	Value
KCS_UNIMP_TOO_MANY_PROFILES	0x4111
KCS_UNIMP_ILLEGAL_TECHNOLOGY	0x4112
Internal	
KCS_INTERNAL_CLASS_CORRUPTED	0x4120
KCS_INTERNAL_DATA_CORRUPTED	0x4121
KCS_PUBLIC_ERRORS_END	0x6FFF
Internal Kodak Errors	
KCS_KODAK_PRIVATE_ERRORS_END	0x7FFE
User Statistics	
KCS_USER_STATUS	0xA000
KCS_USER_STATUS_END	0xFF00
KCS_STATUS_END	KcsForceAlign

Glossary

CMM	Color management module.
icc.ini	File that lists the all the default test scripts packaged with the test suite.
KCMS_PROFILES	Environment variable that specifies the path to the `kcmstest/profiles` directory.
KCMSROOT	Environment variable that specifies the path to the top of the `kcmstest` directory.
kcmstest	A test script interpreter that reads test scripts and performs the KCMS “C” API function calls based upon the commands in the scripts.
load hints	Indicate what, how, when, and where to load and unload a profile.
operation load hints	These are the forward, reverse, simulate, and gamut load hints.
Xstatus	An optional script command keyword that can be used to return an error deliberately generated to test an error condition.