

KCMS™ CMM Reference Manual

SunSoft, Inc.
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.



THE NETWORK IS THE COMPUTER™

Copyright 1997 Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, SunSoft, SunDocs, SunExpress, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. PostScript and Display PostScript are trademarks of Adobe Systems, Inc., which may be registered in certain jurisdictions. KCMS is a trademark of Eastman Kodak Company.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1997 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100 Etatis-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, SunSoft, SunDocs, SunExpress, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc. PostScript et Display PostScript sont des marques d' Adobe Systems, Inc., lesquelles pourront être enregistrées dans des juridictions compétentes. KCMS est une marque déposée d' Eastman Kodak Company.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Contents

Preface.....	xi
New Features.....	xvii
1. KcsShareable Class	1
Protected Members	1
Public Members	2
2. I/O Classes	3
KcsIO Class	3
Enumerations.....	4
Protected Members	4
Public Members	5
Member Function Override Rules.....	6
Example	7
KcsMemoryBlock Class.....	8
Enumerations.....	8
Protected Members	9

Public Members	9
Example	10
KcsFile Class	12
Public Members	13
Examples	13
KcsSolarisFile Class	16
Public Members	17
KcsXWindow Class	18
Public Members	19
Constructing a KcsXWindow Profile Name	19
3. KcsChunkSet Class	21
Protected Members	22
Public Members	22
Example	24
4. KcsLoadable Class	27
Public Members	28
5. KcsProfile Class	31
6. KcsProfileFormat Class	41
Protected Members	42
Public Members	42
Member Function Override Rules	44
External Loadable Interface	45
7. KcsTags Class	47
Public Members	48

8.	KcsXform Class	53
	Enumerations	54
	Protected Members	54
	Public Members	55
	External Loadable Interface	59
	Member Function Override Rules	59
9.	KcsXformSeq Class	61
	Protected Members	62
	Public Members	62
10.	KcsStatus Class	65
	Public Members	66
	External Loadable Interface	67
	Index	69

Tables

Table 1-1	KcsShareable Protected Members	1
Table 1-2	KcsShareable Public Members	2
Table 2-1	KcsIO Enumerations	4
Table 2-2	KcsIO Protected Members.....	4
Table 2-3	KcsIO Public Members	5
Table 2-4	KcsIO Member Function Override Rules	7
Table 2-5	KcsMemoryBlock Enumerations.....	8
Table 2-6	KcsMemoryBlock Protected Members.....	9
Table 2-7	KcsMemoryBlock Public Members.....	9
Table 2-8	KcsFile Public Members	13
Table 2-9	KcsSolarisFile Public Members.....	17
Table 2-10	KcsXWindow Public Members	19
Table 3-1	KcsChunkSet Protected Members	22
Table 3-2	KcsChunkSet Public Members	22
Table 4-1	KcsLoadable Public Members	28
Table 5-1	KcsProfile Protected Members.....	32

Table 5-2	KcsProfile Public Members	34
Table 5-3	KcsProfile Member Function Override Rules	36
Table 6-1	KcsProfileFormat Protected Members	42
Table 6-2	KcsProfileFormat Public Members	42
Table 6-3	KcsProfileFormat Member Function Override Rules....	44
Table 7-1	KcsTags Public Members	48
Table 8-1	KcsXform Enumerations.	54
Table 8-2	KcsXform Enumerations.	54
Table 8-3	KcsXform Public Members	55
Table 8-4	KcsXform External Loadable Interface	59
Table 8-5	KcsXform Member Function Override Rules	59
Table 9-1	KcsXformSeq Protected Members	62
Table 9-2	KcsXformSeq Public Members	62
Table 10-1	KcsStatus Class Public Methods	66
Table 10-2	KcsSolarisFile External Loadable Interface	67

Code Samples

Code Example 2-1	<code>KcsIO</code> Example.....	7
Code Example 2-2	Resizing Memory with <code>KcsMemoryBlock</code>	10
Code Example 2-3	Reading a File From a Specified Offset	13
Code Example 2-4	Writing to a File From the Last Cursor Position.....	14
Code Example 3-1	Getting, Reading and Writing a Chunk.....	24
Code Example 5-1	<code>KcsProfile</code> Class and <code>KcsGetAttribute()</code>	37
Code Example 5-2	<code>KcsProfile</code> Class and <code>KcsConnectProfiles()</code>	38

Preface

The *KCMS CMM Reference Manual* provides detailed descriptions of the Kodak Color Management System (KCMS™) foundation library. This library is a graphics porting interface (GPI) implemented in C++ for creating KCMS color modules. A set of C++ classes are supplied that can be derived from and extended. You can add attributes to the current list, incorporate new color processing technology, or support alternate profile formats.

Use this book with the *KCMS CMM Developer's Guide* which provides an in-depth view of the KCMS framework and how the API works with this GPI, how to derive from each C++ class, how to create a dynamically loadable CMM, and how to add profiles to the system.

Who Should Use This Book

Use this book if you are interested in:

- Writing your own color management module (CMM)
- Creating your own profile format
- Adding attributes to the ICC profile format
- Overriding various class methods

Before You Read This Book

You should be familiar with the Kodak Color Management System (KCMS) API which is part of the SDK; see the *KCMS Application Developer's Guide*.

You should also have an understanding of C++ and Solaris dynamic loading technology. Solaris dynamic loading is discussed in the *Linker and Libraries Guide* and in the following manual pages in `man Pages(1): User Commands` and `man Pages(3): Library Routines`:

- `ld(1)`
- `dlopen(3)`
- `dlclose(3)`
- `dlerror(3)`
- `dlsym(3)`
- `OWconfigInit(3)`
- `OWconfigGetAttribute(3)`
- `OWconfigFreeAttribute(3)`
- `OWconfigClose(3)`

A basic understanding of color science is also assumed; references are included in the Bibliography of the *KCMS Application Developer's Guide*.

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 *Installation Instructions for Solaris 2.6 (SPARC Platform Edition)* and the *Installation Instructions for Solaris 2.6 (Intel Platform Edition)* manuals.

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

How This Book Is Organized

Note – Each chapter in this book describes relevant classes in the KCMS architecture. Although the DDK header files may include additional information (private and methods and other internal interfaces) for each class, be aware that the chapters in this book present all the methods you need to be concerned about to write your CMM.

The chapters are organized as follows:

Chapter 1, “KcsShareable Class,” describes the `KcsShareable` class.

Chapter 2, “I/O Classes,” describes these I/O classes: `KcsIO`, `KcsFile`, `KcsMemoryBlock`, `KcsSolarisFile`, and `KcsXWindow`.

Chapter 3, “KcsChunkSet Class,” describes in detail the `KcsChunkSet` class.

Chapter 4, “KcsLoadable Class,” describes in detail the `KcsLoadable` class.

Chapter 5, “KcsProfile Class,” describes in detail the `KcsProfile` class.

Chapter 6, “KcsProfileFormat Class,” describes in detail the `KcsProfileFormat` class.

Chapter 7, “KcsTags Class,” describes in detail the `KcsAttributeSet` class.

Chapter 8, “KcsXform Class,” describes the member functions in the `KcsXform` class.

Chapter 9, “KcsXformSeq Class,” describes the member functions in the `KcsXformSeq` class.

Chapter 10, “KcsStatus Class,” describes in the `KcsStatus` class.

Related Books

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

- *KCMS Application Developer’s Guide*
- *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>.

Ordering Sun Documents

The SunDocsSM 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™ 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 and Symbols Mean

The following table describes the type changes and symbols used in this book:

Table P-1 Typographic Conventions

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

KcsId Naming Convention

Each class contains a `KcsId` that uniquely identifies that class. Most `KcsIds` are defined in the `kcsids.h` header file. The naming conventions shown in the following table are used for the `KcsId` for each class in the KCMS framework. The `#defines` are helpful in `switch` statements.

Table P-3 `KcsId` Naming Conventions

Item	Convention	Examples
<code>const</code>	<code>Kcs<Base Class Id><Derived Class Id>Id</code>	<code>KcsSharIOId</code>
<code>#define</code>	<code>Kcs<Base Class Id><Derived Class Id>Idd</code>	<code>KcsSharIOIdd</code>

Equivalent Terms In This Book

For historic reasons, this book uses several equivalent Kodak and International Color Consortium (ICC) terms. The terms evolved at different times. Development of the ICC specification introduced new ICC terms with meanings the same as (or similar to) already existing Kodak terms.

You should be familiar with the terms listed in the table below, as you will encounter them in the ICC specification and KCMS color management documentation, as well as in the KCMS header files and example programs. The terms are defined as they are introduced in this book..

Table P-4 Equivalent ICC and Kodak Terms

Kodak Term	ICC Term
attribute	tag
device color profile (DCP)	input, display, or output profile
effects color profile (ECP)	abstract profile
complete color profile (CCP)	device link profile
profile format Id or magic number	profile file signature
reference color space (RCS)	profile connection space (PCS)

Note – The text in this book uses the term *attribute* instead of *tag*, (but code examples and header files may use *tag* for the historic reasons previously mentioned).

New Features

The following information is about features provided in this release of the KCMS product.

KCMS is Multithread Safe

In this release, KCMS supports multithreaded programs.

OWconfig File Modification

The procedure for updating the `OWconfig` file has changed. Using the interactive program called `OWconfig_sample`, you can insert and remove configuration entries in the `OWconfig` file.

KcsShareable Class



This chapter describes the KCMS framework's primary base class, the `KcsShareable` class. This class is at the top of the KCMS class hierarchy. The `KcsShareable` class allows any of its derivatives to be shared.

As you read this chapter, you will find it helpful to have access to the `kcsshare.h` header file.

The constant and `#define` identifiers for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsRlocSharLoadId = {(0x53686172UL)}; /* 'Shar' */  
#define KcsRlocSharLoadIdd (0x53686172UL) /* 'Shar' */
```

The protected and public members are described.

Protected Members

The `KcsShareable` class provides the following protected members.

Table 1-1 KcsShareable Protected Members

Protected Member	Description
<code>virtual ~KcsShareable();</code>	Destructor. A protected member.

Public Members

The `KcsShareable` class provides the following public members.

Table 1-2 `KcsShareable` Public Members

Public Member	Description
<pre>virtual KcsShareable *attach(long howMany = 1, KcsAttachType aAttachFlag = KcsAttMem, KcsStatus *aStatus = NULL);</pre>	Use this method when you want to share an object already allocated. It returns a pointer to the shared object. All shared objects <i>must</i> <code>dettach()</code> instead of using <code>delete</code> . <code>aAttachFlag</code> and <code>aStatus</code> are only used with the <code>KcsLoadable</code> class override.
<pre>void dettach(long howMany = 1, KcsAttachType aDettachFlag = KcsAttMem, KcsStatus *aStatus = NULL);</pre>	Deletes an object. The actual deallocation only happens when no other object is sharing this object. <code>aDettachFlag</code> and <code>aStatus</code> are only used with the <code>KcsLoadable</code> class override.
<pre>static long getGlobalCount();</pre>	Returns the total number of objects sharing any other objects in the system. Use for debugging.
<pre>long getUseCount() {return(useCount);};</pre>	Returns the current number of objects sharing this object. Use for debugging.
<pre>KcsShareable(KcsStatus *status, long nUse = 1);</pre>	Constructor.

This chapter describes the following KCMS input/output (I/O) classes:

- `KcsIO`
- `KcsFile`
- `KcsMemoryBlock`
- `KcsSolarisFile`
- `KcsXWindow`

The `KcsIO` class provides a common interface for I/O operations such as read and write. The `KcsIO` class is a derivative of the `KcsShareable` class (see Chapter 1, “`KcsShareable` Class”). The `KcsFile`, `KcsMemoryBlock`, `KcsSolarisFile`, and `KcsXWindow` are derivatives of the `KcsIO` class. These derivatives provide I/O for more specific types of data storage.

As you read this chapter, you will find it helpful to have access to the following header files:

- `kcsio.h`, `kcsfile.h`, `kcsmbk.h`, `kcssolfi.h`, and `kcsxwin.h`
- `kcsshare.h` and `kcsids.h`

KcsIO Class

With a common interface, the `KcsIO` class maintains device-, platform-, and transport-independent I/O functionality for all derivatives.

The header file for the class is `kcsio.h`. The constant and `#define` identifiers for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsSharIOId = {(0x494F0000UL)}; /* 'IO' */
#define KcsSharIOId (0x494F0000UL) /* 'IO' */
```

The enumerations and protected and public members are described, as well as the member function override rules when deriving from this class.

Enumerations

The `KcsIO` class provides the following enumerations.

Table 2-1 KcsIO Enumerations

Enumeration	Description
<pre>enum KcsIOPosition {KCS_OFS, KCS_BOO, KCS_CUR};</pre>	<p>Used for calls to <code>setCursorPos()</code>. KCS_OFS is relative to beginning of I/O object+baseoffset. KCS_BOO is relative to beginning of the I/O object. KCS_CUR is relative to present I/O cursor.</p>

Protected Members

The `KcsIO` class provides the following protected members.

Table 2-2 KcsIO Protected Members

Protected Member	Description
<pre>KcsStatus aSysError(const char *callersName, const char *sysName, KcsStatus stat, const int sysErrCode);</pre>	<p>A protected member that returns a <code>KcsStatus</code> object that contains the OS error in the <code>causingError</code> data member. Use <code>callersName</code> for debugging; it is recommended that you change to a non-NULL value.</p>

Public Members

The `KcsIO` class provides the following public members.

Table 2-3 `KcsIO` Public Members

Member Function	Description
<pre>virtual KcsStatus absRead(const long index, const long bytesWanted, void *buffer, const char *callersName = NULL);</pre>	Absolute read an I/O object already opened or allocated. Supply a count of number of bytes to read and a buffer.
<pre>virtual KcsStatus absWrite(const long index, const long numBytes2Write, const void *buffer, const char *callersName = NULL);</pre>	Absolute write to some number of bytes in <code>numBytes2Write</code> from the buffer to the data store at the current position of the cursor
<pre>virtual KcsStatus copyData(KcsIO *anotherIO);</pre>	Copies all data in I/O object to anotherIO object.
<pre>static KcsIO * createIO(KcsStatus *aStatus, const KcsProfileDesc *aDesc);</pre>	Static method that creates an I/O object, by calling either a <code>KcsIO</code> derivative constructor within the KCMS library or a run-time loadable constructor.
<pre>virtual KcsStatus getEOF(long *theEOF) = 0;</pre>	Returns the end-of-file (EOF) position.
<pre>virtual long getOffset();</pre>	Returns the permanent offset into the I/O object..
<pre>virtual KcsIOType getType() = 0;</pre>	Returns the type of I/O object.
<pre>virtual int isEqual(KcsIO *anotherIO) = 0;</pre>	Determines if this I/O object and another I/O object are working on the same data stores. The base offsets must also be the same for them to return true.
<pre>KcsIO(KcsStatus *status, const unsigned long absBaseOffset = 0);</pre>	Constructor that initializes the <code>baseOffset</code> data member with the values passed in.
<pre>virtual ~KcsIO();</pre>	Destructor.
<pre>virtual KcsStatus relRead(const long bytesWanted, void *buffer, const char *callersName = NULL) = 0;</pre>	Reads <code>bytesWanted</code> from the I/O object from the current position of the cursor. Positions the cursor after the last byte read.

Table 2-3 KcsIO Public Members (Continued)

Member Function	Description
<pre>virtual KcsStatus relWrite(const long numBytes2Write, const void *buffer, const char *callersName = NULL) = 0;</pre>	Relative write of the number of bytes in numBytes2Write from the buffer to the data store at the current position of the cursor. Positions the cursor after the last byte written.
<pre>virtual KcsStatus replaceData(const unsigned long offset, const unsigned long oldSize, const void *buffer, const unsigned long newSize, const char *callersName = NULL);</pre>	Replaces bytes of different lengths in an I/O object. Specifies where to start writing and size of old and new data. If the old data is longer than the new data, the I/O object is compressed. If the new data is longer than the old data, everything after the old data is moved to the end of the I/O object. If an error occurs, the cursor is where it was before the error occurred.
<pre>virtual KcsStatus setCursorPos(long position, const KcsIOPosition mode= KCS_OFS, const char *callersName = NULL) = 0;</pre>	Sets the I/O object cursor to a caller-supplied position. Positions the I/O object cursor to a specific spot in the object. Mode is defined by the enum KcsFilePosition.
<pre>virtual KcsStatus setEOF(long theEOF) = 0;</pre>	Sets the EOF to a caller-supplied position. If the new EOF is greater than the old EOF, the storage for the new space is undefined.
<pre>virtual void setOffset(long theOffset);</pre>	Sets the base offset to a specified position.

Member Function Override Rules

The following table tells you which KcsIO member functions you must override and can override. The member functions indicated with an “X” in the Must column are required to derive successfully from this base class. Others

can be used and not overridden.

Table 2-4 KcsIO Member Function Override Rules

Member Function	Override Rules	
	Must	Can
getEOF()	X	
getType()	X	
isEqual()	X	
KcsIO()	X	
~KcsIO()		X
relRead()	X	
relWrite()	X	
setCursorPos()	X	
setEOF()	X	
setOffset()		X

Example

The following code shows you how to use a KcsIO derivative as a data member or as an object to pass into a function.

Code Example 2-1 KcsIO Example

```
//Read 4 bytes from KcsIO starting at byte 8.
long getTheSecondLong(KcsStatus *aStat, KcsIO *aIO)
{
    long badNumber = -1;
    long sSecondLong;

    if (aIO == NULL)
        return(badNumber);
    if ((*aStat = aIO->absRead(8, 4, &sSecondLong)) == KCS_SUCCESS)
        return(sSecondLong);
    else
        return(badNumber);
}
```

KcsMemoryBlock *Class*

The `KcsMemoryBlock` class is a memory-based I/O derivative of the `KcsIO` class. It provides a way to manipulate blocks of memory by creating a `KcsIO` object. You can use the protected and public member functions in this class in the implementation of your CMM; you cannot override any member function in this class.

The header file for the class is `kcsmbk.h`.

Note – It is highly recommended that you do *not* use any of the variables and functions for handle-based memory in the `kcsmbk.h` header file. Handle-based memory is not required on the Solaris system.

The constant and `#define` identifiers for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsIOMBlkId = {(0x4D426C6BUL)}; /* 'MBlk' */
#define KcsIOMBlkIdd (0x4D426C6BUL) /* 'MBlk' */
```

In addition to the `KcsIO` virtual member functions that must be overridden for a minimal `KcsIO` implementation, the `KcsMemoryBlock` class has member functions for manipulating blocks of memory. See Table 2-4 on page 7 for a list of the member functions minimally required to derive from the `KcsIO` class.

The enumerations and protected and public members are described.

Enumerations

The `KcsMemoryBlock` class provides the following enumerations.

Table 2-5 `KcsMemoryBlock` Enumerations

Enumeration	Description
<pre>enum KcsMemoryKind { KCS_APPLICATION_HEAP, KCS_SYSTEM_HEAP };</pre>	enum used in <code>setCursorPos()</code> . The object is in the application or system heap.

Protected Members

The `KcsMemoryBlock` class provides the following protected members

Table 2-6 `KcsMemoryBlock` Protected Members

Protected Member	Description
<code>long allocMe;</code>	Determines whether memory block is allocated.
<code>char* curPtr();</code>	Returns the current address of the memory block + position.
<code>long numElements;</code>	Number of elements if memory block is an array.
<code>long pos;</code>	Current position in memory block.
<code>long realEOF;</code>	Number of bytes actually written to memory file.
<code>long size;</code>	Number of bytes allocated to contain memory file.
<code>char *startPtr;</code>	Start of char-pointer-based memory block.

Public Members

The `KcsMemoryBlock` class provides the following public members.

Table 2-7 `KcsMemoryBlock` Public Members

Public Member	Description
<code>KcsStatus get(char* buf, long nbytes);</code>	Gets <code>nbytes</code> of data starting at the current cursor position. Post-increments the cursor position by <code>nbytes</code> .
<code>long getNumElements() {return numElements;}</code>	Returns the number of elements in the <code>KcsMemoryBlock</code> object as a <code>long</code> .
<code>long getSize();</code>	Returns the size of the <code>KcsMemoryBlock</code> object.
<code>KcsMemoryBlock(KcsStatus *status, char *start, long size, long numElements = 1, const unsigned long absBaseOffset = 0);</code>	Constructor. Specifies a character pointer in <code>start</code> , block size in <code>size</code> , and number of elements in <code>numElements</code> .

Table 2-7 KcsMemoryBlock Public Members (Continued)

Public Member	Description
<pre>KcsMemoryBlock(KcsStatus *status, long size, long numElements = 1, const unsigned long absBaseOffset = 0, const KcsMemoryKind kind = KCS_APPLICATION_HEAP);</pre>	<p>Constructor. Allocates and deallocates the memory.</p>
<pre>virtual ~KcsMemoryBlock();</pre>	<p>Destructor.</p>
<pre>long position() {return pos;}</pre>	<p>Returns the current cursor position.</p>
<pre>void position(long posit) {pos=posit;}</pre>	<p>Sets the current cursor position.</p>
<pre>KcsStatus put(char* buf, long nbytes);</pre>	<p>Puts nbytes of data into the KcsMemoryBlock object at the current cursor position. Post-increments the cursor position by nbytes.</p>

Example

This example shows you how to change the size of memory with the KcsMemoryBlock class.

Code Example 2-2 Resizing Memory with KcsMemoryBlock

```
KcsStatus resizeIt()
{
    unsigned long sNewSize = 10;
    KcsStatus sStatus;
    KcsMemoryBlock *memBlock;
    char * buffer = {'a','b','c','d'};

    // create a new KcsMemoryBlock object
    memBlock = new KcsMemoryBlock(&sStatus,4);
    if (sStatus != KCS_SUCCESS)
        return (sStatus);
    // put the four bytes above into the new KcsMemoryBlock
    sStatus = memBlock->put(buffer,4);
    if (sStatus != KCS_SUCCESS)
        return (sStatus);
    // resize the KcsMemoryBlock from 4 to 10
    sStatus = memBlock->reSize(sNewSize);
    //Finished with the data
```

Code Example 2-2 Resizing Memory with KcsMemoryBlock (Continued)

```
delete memBlock;  
return (sStatus);  
}
```

KcsFile *Class*

The `KcsFile` class is a file I/O derivative of the `KcsIO` class. It provides a way to manipulate files by creating a `KcsIO` object. You can use the protected and public member functions in this class in the implementation of your CMM; you cannot override any member function in this class.

The header file for the class is `kcsfile.h`.

Note – It is highly recommended that you do *not* use any of the variables and functions for handle-based memory in the `kcsmbk.h` header file. Handle-based memory is not required on the Solaris system.

The constant and `#define` identifiers of this class as defined in the `kcsids.h` header file are:

```
const KcsId KcsIOFileId = {(0x46696C65UL)}; /* 'File' */
#define KcsIOFileIdd (0x46696C65UL) /* 'File' */
```

In addition to the `KcsIO` virtual functions that must be overridden for a minimal `KcsIO` implementation, the `KcsFile` class has member functions for reading from and writing to a file. See Table 2-4 on page 7 for detailed information on the virtual functions that are minimally required to derive from the `KcsIO` class.

This class does not have any protected members; the public members are described.

Public Members

The `KcsFile` class provides the following public members.

Table 2-8 `KcsFile` Public Members

Member Function	Description
<code>virtual long getFref();</code>	Gets the file description being used.
<code>KcsFile(KcsStatus *status);</code>	Constructor. Creates a file object without a file and offset. You must use <code>setFref()</code> to establish a link to a file before using any other methods.
<code>KcsFile(KcsStatus *status, const KcsFileId anFref, const unsigned long absBaseOffset = 0);</code>	Constructor. Creates a file object with an open file and offset.
<code>virtual ~KcsFile();</code>	Destructor.
<code>virtual void setFref(long theFref);</code>	Sets the file to use and the offset.

Examples

The following examples show you how to use the `KcsFile` class.

Reading a File From a Specified Offset

This example shows you how to read a file from a specified offset with `absRead()`. See the `kcsio.h` header file for a description of `absRead()`.

Code Example 2-3 Reading a File From a Specified Offset

```
KcsStatus readIt()
{
    KcsStatus sStatus;
    KcsFileId sFileRef;
    long index = 32;
    long number;

    // open the file, put the fileRef into sFileRef
    sFileRef = open ("Profile", O_RDWR);
    if (sFileRef == -1)
        return (KCS_IO_ERROR);
    // create a file object
    file = new KcsFile(&sStatus, sFileRef, 0);
    if (sStatus != KCS_SUCCESS)
```

Code Example 2-3 Reading a File From a Specified Offset (Continued)

```

        return(sStatus);

        // using the file object, read from the file into a buffer.
        sStatus = file->absRead(index, sizeof(long), &number);
        delete file;
        close(sFileRef);

        return (sStatus);
    }

```

Writing to a File From the Last Cursor Position

This example shows you how to write to a file with `relWrite()`. See Table 2-3 on page 5 for a full definition of `relWrite()`.

Code Example 2-4 Writing to a File From the Last Cursor Position

```

KcsStatus writeIt()
{
    KcsStatus sStatus;
    KcsFileId sFileRef;
    long nbytes;
    char *buffer;

    // open the file, get a fileRef
    sFileRef = open ("Profile", O_RDWR);
    if (sFileRef == -1)
        return (KCS_IO_ERROR);

    // create a file object
    file = new KcsFile(&sStatus, sFileRef);
    if (sStatus != KCS_SUCCESS)
        return(sStatus);

    // Allocate memory for the buffer, fill it with data.
    // Set nbytes to the length of the buffer.
    if ((buffer = (char*) malloc(nbytes)) == NULL)
        return (KCS_IO_ERROR);
    delete file;
    close(sFileRef);

    // using the file object, write the buffer to the file.
    sStatus = file->relWrite(nbytes, buffer);
}

```

Code Example 2-4 Writing to a File From the Last Cursor Position (Continued)

```
// Free the buffer's memory.  
free (buffer);  
delete file;  
close(sFileRef);  
  
return (sStatus);  
}
```

KcsSolarisFile *Class*

The `KcsSolarisFile` class is a derivative of the `KcsIO` class. It is a Solaris-specific `KcsIO` class that provides member functions that:

- Open a file with a partial name
- Search through a list of known directories
- Check for string endings for filename suffixes (`inp`, `mon`, `out`, and `spc`)
- Access files on a remote machine

Note – The KCMS daemon, `kcms_server`, must be running to access remote files. Remote access is read only. See the `kcms_server(1)` man page.

The `KcsSolarisFile` class creates a pointer to a `KcsFile` or `KcsRemoteFile` object depending on the host location. The derived public methods (`relWrite()`, `relRead()`, `getEOF()`, and `setEOF()`) then call the `KcsIO` pointer to do the actual operation.

The header file for this class is `kcssqlfi.h`.

The `const` and `#define` for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsIOSolfId = {(0x736f6c66UL)}; /* 'solf' */
#define KcsIOSolfIdd (0x736f6c66UL) /* 'solf' */
```

This class does not have any protected members; the public members are described.

Public Members

The `KcsSolarisFile` class provides the following public members.

Table 2-9 `KcsSolarisFile` Public Members

Public Member	Description
<code>virtual KcsIO* getIO();</code>	Returns the I/O pointer.
<code>KcsSolarisFile(KcsStatus *status, const char *filename, const char *hostname, const int oflag, const mode_t mode);</code>	Constructs a new I/O object pointer to a file (full path and suffix not needed) either a remote or local machine. The file is opened with the specified permissions. See the <code>open(2)</code> man page for information on <code>oflag</code> and <code>mode</code> .
<code>virtual ~KcsSolarisFile();</code>	Destructor.

KcsXWindow *Class*

The `KcsXWindow` class is a derivative of the `KcsIO` class. It provides an interface for the X11 Window System connection. It turns X11 information into filenames for access at known directories either on a local or remote system. The KCMS daemon, `kcms_server(1)` must be running to access remote files. Remote access is read only.

The `KcsXWindow` class creates a pointer to a `KcsFile` or `KcsRemoteFile` object depending on the host location which is derived from the X11 Window System information. The derived public members (`relWrite()`, `relRead()`, `getEOF()`, and `setEOF()`) then call the `KcsIO` pointer to do the actual operation.

The header file for the class is `kcsxwin.h`.

The `const` and `#define` for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsIOxwinId = {(0x7877696EUL)}; /* 'xwin' */
#define KcsIOxwinIdd (0x7877696EUL) /* 'xwin' */
```

In addition to the `KcsIO` methods overridden by this class, there are methods for creating filenames remotely or locally with X Window System information. See Table 2-4 on page 7 for detailed information on the virtual functions that are minimally required to derive from the `KcsIO` class.

This class does not have any protected members; the public members are described.

Public Members

The `KcsXWindow` class has the following public members.

Table 2-10 `KcsXWindow` Public Members

Public Member	Description
<pre>virtual KcsIO* getIO();</pre>	Returns the I/O pointer.
<pre>KcsXWindow(KcsStatus *status, const Display *dpy, const int screen, const Visual *visual, const long caller)</pre>	Constructs a new IO object pointer to a profile connected to the machine and display. The specific X Window System profile name is constructed. The location is either a known local directory or a path specified by the <code>KCMS_XTERMINAL_PROFILES</code> environment variable.
<pre>virtual ~KcsXWindow()</pre>	Destructor.

Constructing a `KcsXWindow` Profile Name

The X Window System profiles are created with the KCMS configuration program `kcms_configure`; see the `kcms_configure(1)` on-line man page for more information. The KCMS Calibrator Tool (`kcms_calibrate`) supplies monitor calibration, as well as configuration of the X profiles. See the `kcms_calibrate(1)` man page for more information.

X Window System visual profiles follow this naming convention:

```
<Visual Class><Visual ID in Hex>:<Display #>.<Screen #>
```

For example, for the `PseudoColor` visual on display 0, screen 0, with Visual ID `0x20`, has the following profile name: `PseudoColor0x20:0.0`.

The `Visual ID` is provided with the *visual* argument as well as an indicator to one of the visual names ("`StaticGray`", "`GrayScale`", "`StaticColor`", "`PseudoColor`", "`TrueColor`", or "`DirectColor`").

Similar entries exist for the other visuals. X11 Window System visual profiles are overwritten when a system is recalibrated after setup. The base uncalibrated monitor profiles are in the `/usr/openwin/etc/profiles` directory; therefore, you can always reset the system, if for some reason one of the per-machine profiles is corrupted.

KcsChunkSet *Class*



This chapter describes the KCMS framework `KcsChunkSet` class. You can manage blocks (or chunks) of data arranged in tagged file format with `Chunk` classes. The ICC profile format is directly analogous to the `KcsChunkSet`. The `KcsChunkSet` class is derived from the `KcsShareable` class. See Chapter 1, “`KcsShareable Class`,” for a description of the `KcsShareable` class.

As you read this chapter, you will find it helpful to have access to the following header files:

- `kcsshare.h`
- `kcsio.h` and `kcsmbk.h`
- `kcschu.h` and `kcschunk.h`

Note – It is highly recommended that you do *not* use any of the variables and functions for handle-based memory in these header files. Handle-based memory is not required on the Solaris system.

The header file for the class is `kcschunk.h`. The constant and `#define` identifiers for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsSharChkSid = {(0x43686B53UL)}; /* 'ChkS' */
#define KcsSharChkSidd (0x43686B53UL) /* 'ChkS' */
```

In addition to the `KcsShareable` methods overridden by this class, there are methods for managing chunks of data. The protected and public members are described.

Protected Members

The `KcsChunkSet` class has the following protected members.

Table 3-1 `KcsChunkSet` Protected Members

Protected Member	Description
<pre>virtual KcsStatus readChunkBuffer(const KcsChunkId aChunkId, void *aChunk, unsigned long *aSizeRead, unsigned long aSizeWanted = 0);</pre>	<p>Reads a chunk of bytes indicated by a chunk Id. <code>KCS_CHUNK_ID_ERR</code> is returned if no chunk Ids match in the profile. Assumes the <code>aChunk</code> buffer is allocated. Get the buffer size with <code>getChunkSize()</code> which is defined in Table 3-2.</p>
<pre>virtual KcsStatus writeChunkBuffer(const KcsChunkId aChunkId, void *aChunk, const unsigned long aSize, KcsCompressBoolean aCompress = KcsCompress, KcsLocationEnum aLocationEnum = KcsLocationUnspecified, long aOffset = KcsNoOffsetSpecified);</pre>	<p>Writes a chunk of bytes to a new chunk in the profile, given a <code>void *</code>. Returns a new chunk Id. If a particular location or offset is specified, other chunks are moved to write this chunk at the specified location.</p>

Public Members

The `KcsChunkSet` class has the following public members.

Table 3-2 `KcsChunkSet` Public Members

Public Member	Description
<pre>virtual KcsStatus createChunkId(KcsChunkId *aNewChunkId, unsigned long aChunkSize = 0, KcsLocationEnum aLocationEnum = aKcsLocationUnspecified, long aChunkOffset = KcsNoOffsetSpecified, KcsRegisterBoolean aRegister = KcsIgnoreIdSpecified);</pre>	<p>Allocates <code>aNewChunkId</code> for a chunk introduced to the data type. Puts a new entry into the chunk map. The size of the chunk can be specified, or by default equals 0. The chunk location can be specified in <code>aLocationEnum</code> and <code>aChunkOffset</code> or defaults can be used. Creates specific chunk ids with <code>aRegister</code>.</p>
<pre>virtual KcsStatus deleteChunk(const long aChunkId);</pre>	<p>Deletes the chunk (and its chunk map entry) identified by a <code>chunkId</code>. You are required to check whether any other objects are using this chunk.</p>
<pre>virtual unsigned long getChunkSetLength(KcsStatus *aStatus);</pre>	<p>Returns the length of the domain of the chunk set.</p>

Table 3-2 KcsChunkSet Public Members (Continued)

Public Member	Description
virtual long getChunkSetOffset(KcsStatus *aStatus);	Returns the offset of the beginning of the domain of the chunk set.
virtual unsigned long getChunkSize(KcsStatus *aStatus, KcsChunkId aChunkId);	Returns the chunk size identified by aChunkId. If the chunk is compressed, returns the uncompressed chunk size.
virtual KcsIO * getIO();	Returns the chunk set's I/O object.
virtual KcsChunkId getTopChunkId(KcsStatus *aStatus, unsigned long aSize = 0, long aOffset = KcsNoOffsetSpecified);	Returns the chunk Id of the top chunk which stores information about other chunks. Read this chunk first because it tells what is contained in the other chunks. Or you can specify the size of the chunk and the chunk's offset; by default, aSize = 0 and aOffset = KcsNoOffsetSpecified. If no top chunk exists, call createChunkId().
virtual Boolean isEqual(KcsChunkSet *aNotherCS);	Checks if this chunk set is the same as another chunk set.
KcsChunkSet(KcsStatus *aStatus);	Constructor without an I/O object.
KcsChunkSet(KcsStatus *aStatus, KcsIO *aIoObj, unsigned long aChunkSetLength = KcsUseEOF, KcsSaveBoolean aSaveMapping = KcsSaveChunkSet, int Format = 0);	Constructor of a chunk set instance based on an I/O object. aChunkSetLength can be set to the length of the static store managed by the chunk set if not equal to the length managed by the I/O object. Otherwise, by default, aChunkSetLength is set to the end of the static store managed by the I/O object. When a particular file format does not allow writing out the chunk map, aSaveMapping is False; otherwise, aSaveMapping is, by default, True indicating that the chunk containing the chunk map should be saved.
~KcsChunkSet();	Destructor.
KcsStatus readChunk(const KcsChunkId aChunkId, void *aChunk, unsigned long *aSizeRead, unsigned long aSizeWanted = 0);	Reads a chunk set.
virtual KcsStatus setChunkSetOffset(long aOffset);	Sets aOffset to the beginning of the domain of the chunk set.

Table 3-2 KcsChunkSet Public Members (Continued)

Public Member	Description
virtual KcsStatus setChunkSetPrivateOffset(const long aOffset)	Sets the chunk set's private offset.
virtual KcsStatus setIO(KcsIO *aIoObj);	Sets the chunk set's I/O object.
virtual unsigned long updateChunkUseCount(KcsStatus *aStatus, KcsChunkId aChunkId, long aDelta, unsigned long aComparisonCount);	Calls the ChunkMap object to change the use count of a particular chunk. Compares the use count with an expected value, aComparisonCount.
KcsStatus writeChunk(KcsChunkId aChunkId, void *aChunk, const unsigned long aSize, KcsCompressBoolean aCompress = KcsCompress, KcsLocationEnum aLocationEnum = KcsLocationUnspecified, long aOffset = KcsNoOffsetSpecified);	Writes a chunk set.

Example

The following code shows you some examples of how to use these KcsChunkSet class member functions: getChunkSet(), getChunkSize(), readChunk(), and writeChunk().

Code Example 3-1 Getting, Reading and Writing a Chunk

```

KcsChunkId    myChunkId;
KcsStatus     aStat;
KcsChunkSet*  chunkSet;
char*         mftData;
u_long        mftSize;
u_long        mftWanted;

// Get the chunk data
chunkSet = getChunkSet();
if (chunkSet == NULL)
    return (KCS_INTERNAL_CLASS_CORRUPTED);

//Get the size
mftSize = chunkSet->getChunkSize(&aStat, myChunkId);
if (aStat != KCS_SUCCESS)

```

Code Example 3-1 Getting, Reading and Writing a Chunk (Continued)

```
        return(aStat);

//Make space for the data
mftData = NULL;
if ((mftData = malloc((u_int)mftSize)) == NULL) {
    return (KCS_MEM_ALLOC_ERR);
}

//Read it
aStat = chunkSet->readChunk(myChunkId, mftData, &mftWanted,
                           mftSize);
if (aStat != KCS_SUCCESS)
    return (aStat);

//Zero it out and write it back
memset(mftData, 0, mftSize);
aStat = chunkSet->writeChunk(myChunkId, mftData, mftSize);
if (aStat != KCS_SUCCESS)
    return (aStat);
```


KcsLoadable Class



This chapter describes the `KcsLoadable` class. This base class provides the basic functionality required to create a dynamically loadable CMM. The classes derived from this class (`KcsProfile`, `KcsProfileFormat`, and `KcsXform`) provide functionality to create your own color profiles. The `KcsLoadable` class is derived from the `KcsShareable` class. See Chapter 1, “`KcsShareable Class`,” for a description of the `KcsShareable` class.

As you read this chapter, you will find it helpful to have access to the following header files:

- `kcsload.h` (the `KcsLoadable` class header file)
- `kcsshare.h` and `kcsswap.h`
- `kcschunk.h` and `kcsuidmp.h`

Note – It is highly recommended that you do *not* use any of the variables and functions for handle-based memory in these header files. Handle-based memory is not required on the Solaris system.

The constant and `#define` identifiers for this class are defined in the `kcsload.h` header file as:

```
const KcsId KcsRlocLoadId = {(0x4C6f6164UL)}; /* 'Load' */
#define KcsRlocLoadIdd (0x4C6f6164UL) /* 'Load' */
```

In addition to the `KcsShareable` methods overridden by this class, there are methods for dynamically loading your CMM. The protected and public members are described.

Public Members

The `KcsLoadable` class provides the following public members.

Table 4-1 KcsLoadable Public Members

Public Member	Description
<pre>typedef KcsStatus (*KCS_FUNC_INIT_PTR) (long, long, long *, long *);</pre>	
<pre>virtual KcsShareable * attach(long aHowMany = 1, KcsAttachType aAttachFlag = KcsAttMem, KcsStatus *aStatus = NULL);</pre>	<p>Calls <code>changePermanentUseCount()</code>. If <code>aAttachflag = KcsMemFile</code>, also calls <code>KcsShareable::attach</code>. Returns pointer to the object.</p>
<pre>virtual long changePermanentUseCount(KcsStatus *aStatus, long aDelta);</pre>	<p>Increments or decrements permanent use count of a particular chunk set/chunk Id entry in the unique identifier (UID) map table. Calls <code>KcsChunkSet::updateChunkUseCount()</code> to update the chunk map. Returns new use count.</p>
<pre>static KcsStatus deleteChunkSetsUIDMapEntries(KcsChunkSet *aCS);</pre>	<p>Deletes all the entries in the UID map table associated with a particular chunk set.</p>
<pre>virtual void dettach(long aHowMany = 1, KcsAttachType aDettachFlag = KcsAttMem, KcsStatus *aStatus = NULL);</pre>	<p>If a chunk Id is illegal, searches the UID map table for the chunk Id. Changes permanent use count. If permanent use count == 0, deletes chunk from file. Calls <code>KcsShareable::dettach()</code>.</p>
<pre>virtual KcsChunkId getChunkId(KcsStatus *aStatus, KcsChunkSet *aCS);</pre>	<p>Looks in the UID map table and returns the chunk Id associated with this object and <code>aCS</code>. If no entry is found in the UID map table, it returns <code>KCS_OBJMAP_ENTRY_NOT_FOUND</code> and <code>KcsIllegalChunkId</code>.</p>
<pre>virtual KcsChunkId getChunkId();</pre>	<p>Very useful function; returns the chunk Id portion of the UID associated with this object.</p>
<pre>virtual KcsChunkSet *getChunkSet();</pre>	<p>Very useful function; returns the chunk set portion of the UID associated with this object.</p>
<pre>static KcsLoadable * getObjFromUIDMap(KcsStatus *aStatus, KcsChunkSet *aCS, KcsChunkId aChunkId);</pre>	<p>Checks if object identified by this chunk set and chunk Id is instantiated. If object is in UID map table, it returns a pointer to the object; else it returns <code>NULL</code>.</p>
<pre>virtual KcsStatus isLoadable();</pre>	<p>Returns <code>KcsSuccess</code> if the loadable object can load and regenerate itself.</p>

Table 4-1 KcsLoadable Public Members (Continued)

Public Member	Description
<code>virtual long isLoaded();</code>	Returns non-zero if the state of the loadable object is loaded.
<code>KcsLoadable(KcsStatus *);</code>	Constructor.
<code>KcsLoadable(KcsChunkSet *, KcsChunkId, KcsStatus *);</code>	Constructor that instantiates a loadable based on a UID that is a combination of <code>KcsChunkSet</code> and <code>KcsChunkId</code> .
<code>virtual ~KcsLoadable();</code>	Destructor. Deallocates all resources associated with specified instance.
<code>virtual KcsStatus load(const KcsLoadHints aLoadHints = KcsLoadAllNow, KcsCallbackFunction aCallback = NULL);</code>	Regenerates all necessary state from the static store associated with the <code>aLoadHints</code> .
<code>KcsStatus putObjIntoUIDMap(KcsChunkSet *aCS, KcsChunkId aChunkId);</code>	Puts the object identified by this chunk set and chunk Id in the UID map table. Returns a pointer to the object.
<code>virtual KcsStatus save() {return(KCS_SUCCESS);};</code>	Saves all object state information to its static store.
<code>KcsStatus save(KcsChunkSet *, KcsChunkId);</code>	Saves all object state information to the supplied static store.
<code>KcsStatus setChunkId(KcsChunkId);</code>	Sets the chunk set portion of the object's UID. Changing the value of <code>ChunkId</code> changes the position of this objects regeneration data within the object's static store.
<code>KcsStatus setChunkSet(KcsChunkSet *);</code>	Sets the chunk set portion of the object's UID. Changing the value of the chunk set changes the object's static store.
<code>virtual KcsStatus setUID(KcsChunkSet *aNewChunkSet);</code>	Tries to get a chunk Id corresponding to itself and a chunk set from the UID map table. If no entry, it calls <code>createChunkId()</code> to get a new chunk Id, does a permanent attach, and then calls <code>setChunkSet()</code> and <code>setChunkId()</code> .
<code>virtual KcsStatus unload(KcsLoadHints aLoadHints = (KcsPurgeMemoryNow));</code>	Minimizes the memory requirements of the object by releasing all unnecessary state reclaimed from the static store.

KcsProfile *Class*



This chapter describes the primary `KcsProfile` class. This base class provides basic functionality for using and creating color profiles. It was used to create the KCMS framework API.

As you read this chapter, you will find it helpful to have access to the following header files:

- `kcsprofi.h`
- `kcsfmt.h` and `kcsxform.h`
- `kcsattr.h`, `kcsio.h`, and `icc.h`

The header file for the class is `kcsprofi.h`. The constant and `#define` identifiers for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsLoadProfId = {(0x50726F66UL)}; /* 'Prof' */
#define KcsLoadProfIdd (0x50726F66UL) /* 'Prof' */
```

The protected and public members are described, as well as the member function override rules when deriving from this class.

Protected Members

The `KcsProfile` class provides the following protected members.

Table 5-1 `KcsProfile` Protected Members

Protected Member	Description
<code>KcsProfileFormat *iFormat;</code>	Pointer to a <code>KcsProfileFormat</code> instance.
<code>KcsOperationType *iOpAndCont;</code>	Indicates which operations and content types are supported by the profile. The profile can support the logical OR of any combination of these flags.
<code>virtual KcsStatus createEmptyProfile();</code>	Fill in the minimum amount of data such that when saved and reloaded, no <code>KcsXform</code> instances, and only the minimal tags are needed to load it.
<code>void initDataMembers(void);</code>	Sets all member data to a known state. Allows each constructor to initialize all data members to the empty or initial state.
<code>static long isColorSenseCMM(KcsId);</code>	Do not use this method in new CMMs. It is documented for historic reasons (for existing CMM only).
<code>KcsProfile(KcsStatus *aStat, KcsIO *aIO);</code>	Constructs a <code>KcsProfile</code> based on the static store defined by <code>aIO</code> .
<code>KcsProfile(KcsStatus *aStat, KcsId aCmmId, KcsVersion aCmmVersion, KcsId aProfId, KcsVersion aProfVersion);</code>	Constructs a <code>KcsProfile</code> of the type <code>aCmmId</code> with the version <code>aCmmVersion</code> using a profile format determined by <code>aProfVersion</code> .
<code>KcsStatus setTimeAttribute(KcsAttributeName aAttrName);</code>	Sets attribute specified by <code>aAttrName</code> to current time.

Table 5-1 KcsProfile Protected Members (Continued)

Protected Member	Description
<pre>virtual KcsStatus updateMonitorXforms(KcsCharacterizationData *aChar, KcsCalibrationData *aCal, void *CMMSpecificData, KcsCallbackFunction aCallback);</pre>	<p>Updates the monitor transformations based on aChar and aCal using the CMM's techniques. Returns either KCS_CALIBRATION_UN SUPPORTED or KCS_CHARACTERIZATION_UN SUPPORTED, unless overridden. Some derivatives require specific data in CMMSpecificData. All CMMs do a generic update if CMMSpecificData is set to NULL. Some CMMs callback into aCallback if set to non NULL.</p>
<pre>virtual KcsStatus updatePrinterXforms(KcsCharacterizationData *aChar, KcsCalibrationData *aCal, void *CMMSpecificData, KcsCallbackFunction aCallback);</pre>	<p>Updates the printer transformations based on aChar and aCal using the CMM's techniques. Returns either KCS_CALIBRATION_UN SUPPORTED or KCS_CHARACTERIZATION_UN SUPPORTED, unless overridden. Some derivatives require specific data in CMMSpecificData. All CMMs do a generic update if CMMSpecificData is set to NULL. Some CMMs callback into aCallback if set to non NULL.</p> <p>Note that this function currently is not supported in the KCMS default CMM.</p>
<pre>virtual KcsStatus updateScannerXforms(KcsCharacterizationData *aChar, KcsCalibrationData *aCal, void *CMMSpecificData, KcsCallbackFunction aCallback);</pre>	<p>Updates the scanner transformations based on aChar and aCal using the CMM's technique. Returns either KCS_CALIBRATION_UN SUPPORTED or KCS_CHARACTERIZATION_UN SUPPORTED, unless overridden. Most scanners need both aCal and aChar set to generate valid transforms. Some derivatives require specific data in CMMSpecificData . All CMMs update if CMMSpecificData is set to NULL. Some CMMs callback into aCallback if set to non NULL.</p>

Public Members

The KcsProfile class provides the following public members.

Table 5-2 KcsProfile Public Members

Public Member	Description
<pre>virtual KcsStatus connect(const long count, KcsProfile **sequence, const KcsOperationType opAndHints, KcsProfile **result, long *failingProfileIndex);</pre>	<p>Connects the list of KcsProfile's pointers passed in by the caller via the sequence parameter. Returns the new KcsProfile object result based on that list. failingProfileIndex indicates the number of the profile in the input list that failed. If element 0 in the array caused the connection to fail, 1 is returned.</p>
<pre>static KcsProfile * createProfile(KcsStatus *aStat, aIO *aIO);</pre>	<p>Constructs the correct runtime-loadable or internal KcsProfile based on the static store defined by aIO.</p>
<pre>static KcsProfile * createProfile(KcsStatus *Stat, KcsId aCmmId = KcsProfKCMSId, KcsVersion aCmmVersion = KcsProfKCMSVersionId, KcsId aProfId = Kcs2Id(icMagicNumber), KcsVersion aProfVersion = Kcs2Id(icVersionNumber));</pre>	<p>Constructs the correct runtime-loadable or internal KcsProfile of aCmmId type with the aCmmVersion version using a aProfVersion profile format.</p>
<pre>virtual KcsStatus evaluate(const KcsOperationType opAndHints, KcsPixelLayout source, KcsPixelLayout dest, KcsCallbackFunction progress);</pre>	<p>Transforms the caller's data. Use opAndHints to indicate which transform and content type your 'CMM is providing.</p>
<pre>virtual KcsStatus getAttribute(KcsAttributeName aName, KcsAttributeValue *aValue);</pre>	<p>Sets aValue to the attribute's aName value.</p>
<pre>virtual KcsStatus getAttribute(KcsAttributeName aName, void *data);</pre>	<p>Sets aData to the attribute's aName value.</p>
<pre>virtual KcsProfileFormat * getFormat();</pre>	<p>Gets the profile format.</p>
<pre>virtual KcsOperationType getOpandCont() {return(iOpAndCont)};</pre>	<p>Returns the supported operations and content hints.</p>
<pre>virtual KcsXform * getXform(KcsStatus *status, const KcsXformType XfType);</pre>	<p>Returns the transform of the XfType type.</p>

Table 5-2 KcsProfile Public Members (Continued)

Public Member	Description
<code>virtual KcsStatus isLoadable();</code>	Returns <code>KCS_SUCCESS</code> if meets all loadability requirements.
<code>KcsProfile(KcsStatus *aStat);</code>	Constructor.
<code>virtual ~KcsProfile(void);</code>	Destructor. Frees up accumulated memory by calling all member object's destructors (with <code>delete()</code>).
<code>virtual KcsStatus optimize(const KcsOptimizationType type, KcsCallbackFunction progress);</code>	Makes profile as fast and small as possible. Specify optimization for speed or accuracy with the <code>type</code> parameter.
<code>virtual KcsStatus propagateAttributes2Xforms();</code>	Propagates all the expected attributes from the profile attribute set to the attribute sets of the transforms.
<code>KcsStatus save(KcsIO *);</code>	Saves to the static store indicated by <code>aIO</code> .
<code>virtual KcsStatus setAttribute(KcsAttributeName aName, KcsAttributeType aType, const char *aValue);</code>	Sets the value of <code>aName</code> to <code>aValue</code> . <code>aValue</code> is interpreted as the type <code>aType</code> .
<code>virtual KcsStatus setAttribute(KcsAttributeName aName, const KcsAttributeValue *aValue);</code>	Sets the value of <code>aName</code> to <code>aValue</code> . <code>aValue</code> is interpreted as the default type of <code>aName</code> .
<code>virtual KcsStatus setAttribute(KcsAttributeName aName, void *data);</code>	Sets the value of <code>aName</code> to <code>data</code> interpreted as the default type of <code>aName</code> .
<code>virtual KcsStatus setOpAndCont(KcsOperationType aOpAndCont);</code>	Sets the support operations and content hints to <code>aOpAndCont</code> .

Table 5-2 KcsProfile Public Members (Continued)

Public Member	Description
<pre>virtual KcsStatus setXform(KcsXformType aXformId, KcsXform *aXform);</pre>	<p>Sets the aXformId to aXform. If the transform is null, removes attachment to that particular KcsXform instance, if one exists. Directly implemented by calling KcsProfileFormat::setObject().</p>
<pre>virtual KcsStatus updateXforms(KcsCharacterizationData *aChar, KcsCalibrationData *aCal, void *aCMMSpecificData = NULL, KcsCallbackFunction aCallback = NULL);</pre>	<p>Takes all data supplied and information contained within its KcsAttributeSet instance to determine which type of device to update. Passes aChar, aCal, aCMMSpecificData and aCallback to the appropriate updateDeviceXforms call (where Device is Scanner Monitor Printer). (See Table 5-1 for descriptions of these protected members.)</p>
<pre>long xformIsNOP(const KcsXformType);</pre>	<p>Indicates whether the KcsXformType acts on the data passed to it. If no xform is available, returns true.</p>

Member Function Override Rules

The following table tells you which KcsProfile member functions you must override and can override when deriving from this class. The member functions indicated with an “X” in the Must column are required to derive successfully from this base class. Others may be used but not overridden.

Table 5-3 KcsProfile Member Function Override Rules

Member Function	Override Rules	
	Must	Can
connect()		X
createEmptyProfile()		X
evaluate()		X
getAttribute()		X
initDataMember()		X
isColorSenseCMM()		X
KcsProfile()	X	

Table 5-3 KcsProfile Member Function Override Rules (Continued)

Member Function	Override Rules	
	Must	Can
~KcsProfile()		X
load()		X
optimize()		X
propagateAttributes2Xforms()		X
save()		X
setAttribute()		X
setOpAndCont()		X
setTimeAttribute()		X
setXform()		X
unload()		X
updateMonitorXforms()		X
updatePrinterXforms()		X
updateScannerXforms()		X
updateXforms()		X
XformIsNOP()		X

Examples

The following code sample shows you how to interface with the `KcsProfile` class and its derivatives using the `KcsGetAttribute()` KCMS framework API wrapper function.

Code Example 5-1 KcsProfile Class and KcsGetAttribute()

```
KcsStatusId
KcsGetAttribute(KcsProfileId profile, KcsAttributeName name,
               KcsAttributeValue *value)
{
    VirtualWorld vWorld(true, true);
    KcsProfile * *profiles = 0;

    if (gProfileArray == NULL)
```

Code Example 5-1 KcsProfile Class and KcsGetAttribute() (Continued)

```

// no profiles have been loaded or user has not handled a load
// error correctly
return(KCS_BAD_PROFILE_ID);
KcsStatusId stat;
profiles = (KcsProfile * *)KcsLockHandle(gProfileArray);
if (isValid(profile, profiles))
    stat = profiles[profile]->getAttribute(name, value);
else
    stat = KCS_BAD_PROFILE_ID;
KcsUnlockHandle(gProfileArray);
return(stat);
}

```

The following code sample shows you how to interface with the KcsProfile class and its derivatives using the KcsConnectProfiles() KCMS framework API wrapper function.

Code Example 5-2 KcsProfile Class and and KcsConnectProfiles()

```

KcsStatusId
KcsConnectProfiles(KcsProfileId *resultProfileId,
    unsigned long profileCount, KcsProfileId *profileSequence,
    KcsOperationType operationLoadSet,
    unsigned long *failedProfileIndex)
{
    VirtualWorld vWorld(true, true);
    KcsProfile * *profiles = 0;
    KcsStatus result;
    long i;

    if (gProfileArray == NULL)
        // no profiles have been loaded or user hasn't handled a load
        // error correctly
        return(KCS_BAD_PROFILE_ID);

    result = getNewValidIndex(resultProfileId);
    if (result != KCS_SUCCESS)
        return(result);
    profiles = (KcsProfile * *)KcsLockHandle(gProfileArray);
    KcsMemoryBlock * memblk = new KcsMemoryBlock(&result,
        sizeof(KcsProfile * ), profileCount);
    if (memblk == NULL)
        return(KCS_MEM_ALLOC_ERROR);
    KcsProfile * *moreProfiles = (KcsProfile * *)memblk->lock();
}

```


Code Example 5-2 KcsProfile Class and KcsConnectProfiles() (Continued)

```
KcsProfile * madeProfile = NULL;
long failingNum = 0;
for (i = 0; i < profileCount; i++) (
    if (!isValid(profileSequence[i], profiles)) {
        *failedProfileIndex = i;
        *resultProfileId = BAD_PROFILE_ID;
        memblk->unlock();
        memblk->dettach();
        KcsUnlockHandle(gProfileArray);
        // also have to do a detach here !!
        return(KCS_BAD_PROFILE_ID);
    }
    else
        moreProfiles[i] =
            (KcsProfile *)profiles[profileSequence[i]]->attach();
}
result = profiles[profileSequence[0]]->connect(profileCount,
    moreProfiles, operationLoadSet, madeProfile, &failingNum);
*failedProfileIndex = failingNum;
if (result == KCS_SUCCESS)
    profiles[*resultProfileId] = madeProfile;

for (i = 0; i < profileCount; i++) {
    moreProfiles[i]->dettach();
}
memblk->unlock();
memblk->dettach();
KcsUnlockHandle(gProfileArray);

gNumProfilesAllocated++;
return(result);
}
```


KcsProfileFormat *Class*



This chapter describes the KCMS framework `KcsProfileFormat` class. With the `KcsProfileFormat` class interface you can map a profile from a static store into the traditional objects that make up a profile. The `KcsProfileFormat` class is derived from the `KcsLoadable` class. See Chapter 4, “`KcsLoadable` Class,” for a description of the `KcsLoadable` class.

As you read this chapter, you will find it helpful to have access to the following header files:

- `kcsfmt.h`
- `kcsload.h` and `kcsxform.h`
- `kcsattr.h`

The header file for the class is `kcsfmt.h`. The constant and `#define` identifiers for this class are defined in the `kcsids.h` header file as:

```
const KcsId KcsLoadPfmtId = {(0x50666D74UL)}; /* 'Pfmt' */
#define KcsLoadPfmtIdd (0x50666D74UL) /* 'Pfmt' */
```

In addition to the `KcsLoadable` methods overridden by this class, there are methods for mapping data in static store into profiles comprised of objects. The protected and public members are described, as well as the member function override rules when deriving from this class.

Protected Members

The `KcsProfileFormat` class has the following protected members.

Table 6-1 `KcsProfileFormat` Protected Members

Protected Member	Description
<code>virtual KcsStatus deleteXform(KcsXformType aXfType);</code>	Deletes the transforms.
<code>KcsProfileFormat(KcsStatus *aStat, KcsIO *KcsIO, int Format = 0);</code>	Constructor that takes a <code>KcsIO</code> object to construct the appropriate profile format.
<code>KcsProfileFormat(KcsStatus *aStat, KcsId aCmmId, KcsVersion aCmmVersion, KcsId aProfId, KcsVersion aProfVersion);</code>	Constructor. Takes a profile version and creates a blank profile format.
<code>virtual KcsStatus loadObjectMap(const KcsLoadHints aHints = KcsLoadAllNow, KcsCallbackFunction aCallback = NULL);</code>	Loads the profile format's objects mapped to chunk Ids.
<code>virtual KcsStatus saveObjectMap();</code>	Saves profile format's objects mapped to chunk Ids.

Public Members

The `KcsProfileFormat` class has the following public members.

Table 6-2 `KcsProfileFormat` Public Members

Public Member	Description
<code>static KcsProfileFormat * createProfileFormat(KcsStatus *aStat, KcsIO *aIO);</code>	Creates a profile format object from a <code>KcsIO</code> object.
<code>static KcsProfileFormat * createProfileFormat(KcsStatus *aStat, KcsId aCmmId, KcsVersion aCmmVersion, KcsId aProfId, KcsVersion aProfVersion);</code>	Creates a profile format object from CMM and profile information.
<code>virtual void dirtyAttrCache();</code>	Caches an attribute that has changed.
<code>virtual void dirtyXformCache(KcsXformType aXformId);</code>	Caches an Xform that has changed.

Table 6-2 KcsProfileFormat Public Members (Continued)

Public Member	Description
<code>KcsLoadHints generateLoadWhat(const KcsXformType aXfType) const;</code>	Returns the load hint associated with this <code>KcsXformType</code> .
<code>virtual KcsAttributeSet * generateXformAttributes(KcsStatus *aStat, KcsXformType aXfType);</code>	Generates attributes associated with a <code>KcsXform</code> .
<code>virtual KcsStatus getCmmId(KcsId *);</code>	Returns a CMM Id.
<code>virtual KcsStatus getObject(KcsAttributeSet **aAttr, KcsCallbackFunction aFunc = NULL);</code>	Gets the <code>KcsAttributeSet</code> of the format.
<code>virtual KcsStatus getObject(KcsXform **aXform, KcsXformType aXformId, KcsCallbackFunction aFunc = NULL);</code>	Returns the <code>KcsXform</code> that represents the <code>aXformId</code> operation.
<code>virtual long getSaveSize();</code>	Returns the number of bytes to be saved.
<code>static KcsId getTheCMMId(KcsStatus *aStat, KcsIO *aIO);</code>	Gets the CMM identifier for the <code>KcsIO</code> object.
<code>static KcsVersion getTheCMMVersion(KcsStatus *aStat, KcsIO *aIO);</code>	Gets the CMM version for the <code>KcsIO</code> object.
<code>static KcsId getTheProfileFormat(KcsStatus *aStat, KcsIO *aIO);</code>	Gets the profile format using the <code>KcsIO</code> object.
<code>static KcsVersion getTheProfileVersion(KcsStatus *aStat, KcsIO *aIO);</code>	Gets the profile version using the <code>KcsIO</code> object.
<code>virtual long getSaveSize();</code>	Returns the number of bytes needed to save the profile.
<code>virtual KcsStatus initEmptyFormat(KcsIdent aCMMId = KcsKodakColorSenseCMM);</code>	Initializes profile's static store to an initial state.
<code>virtual KcsStatus isSupported(KcsLoadHint aHints);</code>	Returns whether the object(s) associated with <code>aHints</code> is supported by this profile instance.
<code>KcsProfileFormat(KcsStatus *aStat);</code>	Constructor.
<code>virtual ~KcsProfileFormat(void);</code>	Destructor.

Table 6-2 KcsProfileFormat Public Members (Continued)

Public Member	Description
virtual KcsStatus postAttrCompose(KcsTags *aOwningAttrSet, const long aCount, KcsTags **aOrigSequence);	Builds ICC profile sequence description tag.
virtual KcsStatus save();	Saves all objects associated with this profile format.
virtual KcsStatus saveNew(KcsChunkSet *);	Saves using a different chunk set .
virtual KcsStatus setCmmId(KcsId);	Sets the CMM Id.
virtual KcsStatus setObject(KcsAttribute *aAttr);	Sets KcsAttribute to the set of attributes to save in this profile's format.
virtual KcsStatus setObject(KcsXformType aXformId, KcsXform *aXform);	Sets the KcsXformType to the KcsXform class aXform.
virtual KcsStatus unloadWhenMatch(KcsLoadHints aMatchHints = KcsUnloadAfterUse);	Unloads the profile objects associated with aMatchHints, if they match the last hints supplied to unload(). Unloads the correct data automatically. Helps recover memory (call it with KcsUnloadWhenNecessary()).

Member Function Override Rules

The following table tells you which KcsProfileFormat member functions you must override and can override when deriving from this class. The member functions indicated with an “X” in the Must column are required to successfully derive from this base class. Others can be used but not overridden.

Table 6-3 KcsProfileFormat Member Function Override Rules

Member Function	Override Rules	
	Must	Can
deleteXform()		X
dirtyAttrCache()		X
dirtyXformCache()		X

Table 6-3 KcsProfileFormat Member Function Override Rules (Continued)

Member Function	Override Rules	
	Must	Can
generateLoadWhat()		X
generateXformAttributes()		X
getObject()		X
getSaveSize()		X
initEmptyFormat()		X
isSupported()		X
KcsProfileFormat()	X	
~KcsProfileFormat()		X
load()		X
loadObjectMap()		X
postAttrCompose()		X
save()		X
saveNew()		X
saveObjectMap()		X
setCmmId()		X
setObject()		X
unload()		X
unloadWhenMatch()		X

External Loadable Interface

All `KcsProfileFormat` derivatives support runtime derivability. The base class supports the static `createX(UID)` method like many of the other classes in the KCMS framework. It also supports

`createProfileFormat(KcsVersion)` for creation of new profile formats.

```
static KcsProfileFormat *createProfileFormat(  
    KcsStatus *aStat, KcsIO *aIO);  
// Create a profile format given a chunkSet that has  
// already been created.  
static KcsProfileFormat *createProfileFormat(  
    KcsStatus *aStat, KcsId aCmmId, KcsVersion aCmmVersion,  
    KcsId aProfId, KcsVersion aProfVersion);  
// Create a new profile format of a given version.
```

See the *KCMS CMM Developer's Guide* for more information on creating a `KcsProfileFormat` runtime derivative.

KcsTags *Class*



This chapter describes the `KcsTags` class of the KCMS framework. This class provides numerous data members used throughout the KCMS framework and for use in your CMM.

Note – `KcsAttributeSet` is an alias to the `KcsTags` class. This is for historical reasons only.

As you read this chapter, you will find it helpful to have access to the `kcstags.h` header file.

Note – It is highly recommended that you do *not* use any of the variables and functions for handle-based memory in these header files. Handle-based memory is not required on the Solaris system.

The `#define` identifiers for this class are defined in the `kcstags.h` header file as:

```
#define KcsAttributeSet KcsTags
```

In addition to the `KcsLoadable` methods overridden by this class, this class includes data members used throughout the KCMS framework and for use in your CMM. There are no protected members. Public members are described in detail.

Public Members

The `KcsTags` class provides the following public members.

Table 7-1 `KcsTags` Public Members

Public Member	Description
<code>KcsAttributeSet *copy(KcsStatus *status);</code>	A non-shared copy of the object.
<code>static KcsTags *createAttributeSet(KcsStatus *status);</code>	Creates a blank attribute set.
<code>static KcsTags *createAttributeSet(KcsChunkSet *ChunkSet, KcsStatus *status);</code>	Creates an attribute set from a chunk set.
<code>static KcsTags *createAttributeSet(KcsStatus *aStat, KcsChunkSet *aCS, KcsChunkId aChunkId);</code>	Creates an attributes set from a chunk set and chunk Id.
<code>KcsStatus getAttribute(KcsAttributeName tag, KcsAttributeValue *value, KcsAliasUsage aAliasUse = KcsIndirectThroughAlias);</code>	Given a valid attribute, loads the value.
<code>static KcsStatus getAttributeInfo(KcsAttributeName aTag, KcsAttributeType *aType, unsigned long *count, unsigned long *sizeofType, KcsTags *tags = NULL);</code>	Given a valid <code>aTag</code> , populates the supplied parameters with the type and the count (the number of tokens that make up the attribute) of the attribute associated with the supplied <code>aTag</code> .
<code>static unsigned long getAttrSize(KcsAttributeType aTagType);</code>	Returns the size in bytes of the attribute type.
<code>static unsigned long getAttrSize(KcsAttributeValue *aVal);</code>	Returns the attribute size.
<code>static long getICProfSeqDescInfo(icProfileSequenceDesc *aDesc, icDescStruct **aDescPtrs, long nPtrs);</code>	Returns information about an ICC profile sequence description.

Table 7-1 KcsTags Public Members (Continued)

Public Member	Description
<pre>static long getICTextDescInfo(icTextDescription *aDesc, long *aASCIILength = NULL, icUInt8Number **aASCIIPtr = NULL, icInt32Number *aUnicodeCode = NULL, long *aUnicodeLength = NULL, icInt8Number **aUnicodePtr = NULL, icInt16Number *aScriptCodeCode = NULL, long *aScriptCodeLength = NULL, icInt8Number **aScriptCodePtr = NULL);</pre>	Given an ICC text description, returns the parameters contained in it.
<pre>static long KcsTags::getICTextDescSize(long aASCIILength, long aUnicodeLength, long aScriptCodeLength);</pre>	Returns the size of an ICC text description attribute.
<pre>static long getICUcrBgCounts(icUcrBg *aUcrBg, icUInt32Number *aUcrCount = NULL, icUInt32Number *aBgCount = NULL);</pre>	Returns the number of Ucrdatasets in an ICC Ucr Bg curve.
<pre>unsigned long getLargestAttrValSize();</pre>	Returns the size of the largest attribute.
<pre>KcsStatus getTag(long ordinal, KcsAttributeName *aTag);</pre>	Sets the parameter aName to position ordinal in the internal attribute array pointed to by the member handle tagHandle.
<pre>virtual Boolean isReadOnly(KcsAttributeName name);</pre>	Returns True if an attribute is read only. Read-only attributes are: icSigNumTag, icSigListTag, KcsAttrNumber, KcsAttrAttributesSet, and KcsPixelLayoutSupported.
<pre>KcsStatus KcsAttrCompose(const long count, KcsTags **sequence);</pre>	Given a count and a sequence of KcsAttributeSet objects, combines the attributes according to the rules in the current KcsAttributeSet object and populates its attributes accordingly. See the ICC profile format specification for the profile sequence attribute.
<pre>typedef enum { /* ... */ } KcsAttribute;</pre>	ICC attributes.

Table 7-1 KcsTags Public Members (Continued)

Public Member	Description
KcsStatus KcsClassify(const long count, KcsTags **sequence, KcsAttributeName tag);	Attribute composition rule. Classifies the attributes.
KcsStatus KcsCommon(const long count, KcsTags **sequence, KcsAttributeName tag);	Attribute composition rule. Returns status on whether there are common attributes in two profiles.
KcsStatus KcsConcatenate(const long count, KcsTags **sequence, KcsAttributeName tag);	Attribute composition rule. Concatenates attributes.
KcsStatus KcsNever(const long count, KcsTags **sequence, KcsAttributeName tag);	Attribute composition rule. Attribute composition never propagates.
KcsTags(KcsChunkSet *ChunkSet, KcsChunkId chunkId, KcsStatus *status));	Allows you to create an attribute's object from a chunk. A chunk constructor.
KcsTags(char *buffer, unsigned long sizeOfBuf, KcsStatus *status);	Allows a user to create an attribute's object from data found in the character buffer supplied as an input parameter. A character buffer constructor.
KcsTags(KcsStatus *status);	Constructor that allows a user to create an empty attributes object.
~KcsTags();	Destructor.
KcsStatus KcsUseCSInRight(const long count, KcsTags **sequence, KcsAttributeName tag);	Composition rule.
KcsStatus KcsUseLeft(const long count, KcsTags **sequence, KcsAttributeName tag);	Composition rule.

Table 7-1 KcsTags Public Members (Continued)

Public Member	Description
KcsStatus	Composition rule.
KcsUseRight(const long count, KcsTags **sequence, KcsAttributeName tag);	
long returnCurrentNumberOfAttributes(void);	Returns the current number of attributes stored in the attribute array.
KcsStatus save();	Saves the attributes.
KcsStatus saveTags();	Saves the attributes.
KcsStatus setAttribute(KcsAttributeName aName, void *data);	Stores an association of name-to-attribute in the attribute array.
KcsStatus setAttribute(KcsAttributeName tag, KcsAttributeType aType, void *data);	Stores an association of tag-to-attribute in the attribute array.
KcsStatus setAttribute(KcsAttributeName tag, KcsAttributeType aType, const char *value);	Stores an association of tag-to-attribute in the attribute array. The information is supplied in a character string.
KcsStatus setAttribute(KcsAttributeName tag, const KcsAttributeValue *value, KcsAliasUsage aAliasUse = KcsIndirectThroughAlias);	Stores an association of tag-to-attribute in the attribute array. If you use the KcsAttributeValue structure as a parameter and pass in NULL as its value, the attribute will be deleted if it is stored; otherwise an error is returned.
static long KcsTags::setICTextDesc(icTextDescription *aDesc, long aSize, icUInt32Number aASCIILength = 1, icUInt8Number *aASCIIPtr = NULL, icUInt32Number aUnicodeCode = 0, icUInt32Number aUnicodeLength = 1, icUInt8Number *aUnicodePtr = NULL, icUInt16Number aScriptCodeCode = 0, icUInt8Number aScriptCodeLength = 1, icUInt8Number *aScriptCodePtr = NULL);	Puts the various arguments into an ICC text description structure.

KcsXform *Class*



This chapter describes the KCMS framework `KcsXform` class. This base class provides an interface for component transformations of different transformation types; for example, matrix and grid-table based.

As you read this chapter, you will find it helpful to have access to the following header files:

- `kcsxform.h`
- `kcschunk.h` and `kcsload.h`
- `kcsattr.h` and `icc.h`

The header file for the class is `kcsxform.h`. The constant and `#define` identifiers for this class are defined in the `kcsxform.h` header file as:

```
const KcsId KcsLoadXfrmId = {(0x5866726dUL)}; /* 'Xfrm' */
#define KcsLoadXfrmIdd (0x5866726dUL) /* 'Xfrm' */
```

In addition to the `KcsLoadable` methods overridden by this class, this class includes methods for component transformations of different transformation types. The protected and public members are described, as well as the member function override rules when deriving from this class.

Enumerations

The `KcsXform` class has the following enumerations.

Table 8-1 `KcsXform` Enumerations

Enumeration	Description
<code>enum KcsInOut {In, Out};</code>	Enumeration used in <code>getNumComponents()</code> , <code>getComponentDepth()</code> , <code>setNumComponents()</code> , and <code>setComponentDepth()</code> .

Protected Members

The `KcsXform` class has the following protected members.

Table 8-2 `KcsXform` Enumerations

Enumeration	Description
<code>long myChunkId;</code>	Current chunk Id.
<code>KcsOperationType myOpsAndHints;</code>	Current operations and hints.
<code>KcsTransformKind myKindOf;</code>	Current transform.
<code>int callbackInterval;</code>	Rows per callback.

Public Members

The `KcsXform` class has the following public members.

Table 8-3 `KcsXform` Public Members

Public Member	Description
<pre>int areLayoutsCloseEnough(KcsPixelFormat *in1, KcsPixelFormat *in2, KcsPixelFormat *out);</pre>	Returns nonzero if the <i>n</i> th pixel in <i>in1</i> and <i>in2</i> uses the pixel in <i>out</i> without corrupting the (<i>n</i> +1)th pixel of <i>out</i> , (for example, <code>planar<->chunky</code> fails while <code>chunky<->chunky</code> does not); else returns zero.
<pre>int areLayoutsEqual(KcsPixelFormat *l1, KcsPixelFormat *l2);</pre>	Returns non zero if the pixel layout structs in <i>l1</i> and <i>l2</i> are identical (especially the pointers to the data buffers); else returns zero.
<pre>virtual KcsStatus compose(KcsXformSeq *xformSeq, KcsXform **newXform, KcsCallbackFunction progress);</pre>	Generates a new Xform from <i>xformSeq</i> supplied. Connects <i>numXforms</i> transforms supplied in a <code>KcsXformSeq</code> and returns a <code>KcsXform</code> in <i>newXform</i> that has the same effect as the list of Xforms in the order supplied. An error in connecting from <code>techs[n]</code> to <code>techs[n+1]</code> is reflected in <code>errors[n]</code> . A NULL <code>KcsXform</code> is returned on error.
<pre>virtual KcsStatus connect(const KcsOperationType opsAndHints, const long numXforms, KcsXform **technologies, KcsXform **newXform, KcsCallbackFunction progress, long *numberThatFailed);</pre>	Connects <i>numXforms</i> transforms supplied in <i>technologies</i> and returns a <code>KcsXform</code> that has the same effect as the list of Xforms in the order supplied. If an error is found, it is reported. An error in connecting from <code>techs[n]</code> to <code>techs[n+1]</code> is reflected in <code>errors[n]</code> . A NULL <code>KcsXform</code> is returned on error.
<pre>virtual KcsStatus connectSink(KcsPixelFormat *sinkLayout);</pre>	Associates <i>sinkLayout</i> as the default layout buffer for output from this transform.
<pre>virtual KcsStatus connectSource(KcsPixelFormat *sourceLayout);</pre>	Associates <i>sourceLayout</i> as the default layout buffer for input to this transform.
<pre>virtual KcsXform *convertXform(KcsStatus *aStat, KcsId aXformType);</pre>	Converts the instance into a derivative of type <i>aXformType</i> and returns a pointer to the new transform. Does automatic xform conversion when needed.

Table 8-3 KcsXform Public Members (Continued)

Public Member	Description
<pre>static KcsXform * createXform(KcsStatus *status, KcsChunkSet *aChunkSet, KcsChunkId chunkId, KcsAttributeSet *aAttrSet=NULL);</pre>	Creates a KcsXform from a chunk set.
<pre>KcsStatus eval();</pre>	Evaluates the data described in the local <code>inBuffer</code> set by <code>connectSource()</code> through this transform into the buffer described by the <code>outBuffer</code> set by <code>connectSink()</code> . The operation is from <code>myOptType</code> . This provides no progress.
<pre>KcsStatus eval(const KcsOperationType opsAndHints, long *in32, long *out32);</pre>	Evaluates the data described in one 32-bit integer.
<pre>KcsStatus eval(const KcsOperationType opsAndHints, KcsPixelLayout *inBuffer, KcsPixelLayout *outBuffer, KcsCallbackFunction progress) = 0;</pre>	Evaluates the data described in <code>inBuffer</code> through the Xform described by this instance into the buffer described by <code>outBuffer</code> .
<pre>KcsStatus eval(const KcsOperationType opsAndHints, const float **inComp, const float **outComp, const long *inStride, const long *outStride, const long num, KcsCallbackFunction progress);</pre>	Evaluates the data described in the float-based buffer supplied through this transform into the float-based buffer supplied.
<pre>KcsStatus eval(const KcsOperationType opsAndHints, const unsigned char **inComp, const unsigned char **outComp, const long *inStride, const long *outStride, const long num, KcsCallbackFunction progress);</pre>	Evaluates the data described in the byte-based buffer supplied through this transform into the byte-based buffer supplied.
<pre>virtual KcsAttributeSet *getAttrSet(KcsStatus *aStat = NULL);</pre>	Returns the set of attributes associated with this KcsXform instance. If NULL, creates an empty one.
<pre>virtual long getComponentDepth(KcsInOut whichOne);</pre>	Retrieves the component depth.

Table 8-3 KcsXform Public Members (Continued)

Public Member	Description
virtual KcsStatus getLoadOrder(long aNumTypes, KcsLoadSaveSet aAvailableTypes, KcsLoadSaveSet *aOrderOfTypes);	Given aAvailableTypes, returns the order to load the types in an array aOrderOfTypes[aNumTypes].
virtual long getNumComponents(KcsInOut whichOne);	Retrieves the number of components.
KcsOperationType getOpsAndHints();	Retrieves the ops and hints.
virtual KcsStatus getSaveOrder(long aNumTypes, KcsLoadSaveSet aAvailableTypes, KcsLoadSaveSet *aOrderOfTypes);	Given aAvailableTypes, returns the order to save the types in an array aOrderOfTypes[aNumTypes].
static KcsId getXformType(KcsChunkSet *aChunkSet, KcsChunkId chunkId, KcsStatus *stat, int *InComp, int *OutComp);	Gets the Xform type.
typedef long KcsLoadSaveSet;	Used in getLoadOrder() and getSaveOrder().
typedef unsigned long KcsTransformKind;	Used in kindOfTransform().
KcsXform(KcsStatus *status, KcsAttributeSet *aAttrSet=NULL);	Constructor.
KcsXform(KcsStatus *status, KcsOperationType opsAndHints, KcsChunkSet *aChunkSet, KcsChunkId chunkId, KcsAttributeSet *aAttrSet=NULL);	Constructor.
virtual ~KcsXform();	Destructor.
virtual KcsTransformKind kindOfTransform();	Retrieves the kind of transform.
virtual KcsStatus loadU(KcsMemoryBlock *aXform, KcsCallbackFunction aCallback = NULL);	Loads the Xform from the universal buffer supplied.
virtual int numberOfCallbacks(KcsPixelLayout *aIn, KcsPixelLayout *aOut);	If the Xform evaluated data with the supplied pixel layouts and a callback was supplied, tells how many times the callback would be called. The default is once for every four scan lines of aIn.
KcsStatus optimize();	Optimizes the Xform.

Table 8-3 KcsXform Public Members (Continued)

Public Member	Description
virtual KcsStatus optimize(const KcsOperationType opsAndHints, const KcsOptimizationType optimization, KcsCallbackFunction progress);	Optimizes the Xform.
virtual KcsStatus saveU(KcsMemoryBlock *aXform, KcsCallbackFunction aCallback=NULL);	Saves the Xform into the universal buffer supplied.
virtual KcsStatus setAttrSet(KcsAttributeSet *aAttrSet);	Sets the set of attributes associated with this KcsXform instance.
virtual KcsStatus setCallbackInterval(int callbackInt);	Sets the callback interval.
virtual KcsStatus setComponentDepth(KcsInOut whichOne, long Depth = 8);	Sets the component depth.
virtual KcsStatus setDefaultAttributes(void);	Sets up the default set of attributes for Xforms.
virtual KcsStatus setNumComponents(KcsInOut whichOne, long numComp = 3);	Sets the number of components.
void setOpsAndHints(KcsOperationType);	Sets the ops and hints.
KcsStatus validateLayouts(KcsPixelFormat *sourceLayout, KcsPixelFormat *sinkLayout);	Makes sure the source layout and sink layout are compatible with each other and with transform. The color space of the buffer is validated once that field is added to the KcsPixelFormat structure.

External Loadable Interface

Use these `KcsXform` external entry points to load your derivatives at runtime. See the DDK manual *KCMS CMM Developer's Guide* for more information on creating a `KcsXform` derivative as a runtime loadable CMM.

Table 8-4 `KcsXform` External Loadable Interface

Extern "C"	Description
<code>extern long KcsDLOpenXfrmCount;</code>	Holds a counter for the number of times this dynamically loadable module has been loaded.
<code>KcsStatus KcsInitXfrm(long libMajor, long libMinor, long *myMajor, long *myMinor);</code>	Returns <code>KCS_SUCCESS</code> .
<code>KcsXfrm * KcsCreateXfrm(KcsStatus *aStat, KcsChunkSet *aChunkSet, KcsChunkId aChunkId, KcsAttributeSet *aAttrSet);</code>	Returns a <code>KcsXform</code> object by invoking the <code>KcsXform</code> constructor.
<code>KcsStatus KcsCleanupXfrm();</code>	Returns <code>KCS_SUCCESS</code> .

Member Function Override Rules

The following table tells you which `KcsXform` member functions you must override and can override when deriving from this class. The member functions indicated with an "X" in the Must column are required to derive successfully from this base class. Others can be used and not overridden.

Table 8-5 `KcsXform` Member Function Override Rules

Member Function	Override Rules	
	Must	Can
<code>compose()</code>		X
<code>connect()</code>		X
<code>connectSink()</code>		X
<code>connectSource()</code>		X
<code>connectXform()</code>		X

Table 8-5 KcsXform Member Function Override Rules (Continued)

Member Function	Override Rules	
	Must	Can
convertXform()		X
eval()	X	
getAttrSet()		X
getLoadOrder()		X
getSaveOrder()		X
KcsXform()	X	
~KcsXform()		X
loadU()		X
numberOfCallbacks()		X
optimize()		X
saveU()		X
setAttrSet()		X
setCallbackInterval()		X
setComponentDepth()		X
setDefaultAttributes()		X
setNumComponents()		X
validateLayouts()		X

KcsXformSeq *Class*



This chapter describes the KCMS framework `KcsXformSeq` class. This class provides an interface to manipulate a list or concatenation of `KcsXforms` in a sequence as one `KcsXform` instance.

As you read this chapter, you will find it helpful to have access to the following header files:

`kcsxfseq.h` and `kcsxform.h`

Note – It is highly recommended that you do *not* use any of the variables and functions for handle-based memory in these header files. Handle-based memory is not required on the Solaris system.

The header file for the class is `kcsxfseq.h`. The constant and `#define` identifiers for this class are defined in the `kcsxfseq.h` header file as:

```
const KcsId KcsXfrmSeqId = {0x53657120}; /* 'Seq ' */
#define KcsXfrmSeqIdd (0x53657120) /* 'Seq ' */
```

In addition to the `KcsLoadable` and `KcsXform` methods overridden by this class, there are methods to manipulate a list or concatenation of `KcsXforms`. The protected and public members are described.

Protected Members

The `KcsXformSeq` class has the following protected members.

Table 9-1 `KcsXformSeq` Protected Members

Protected Member	Description
<pre>virtual KcsStatus evalPrep(unsigned long *seqCount, KcsPixelFormat *inBuffer, KcsPixelFormat *outBuffer, KcsPixelFormat *genIn, KcsPixelFormat *genOut, KcsPixelFormat *remIn, KcsPixelFormat *remOut);</pre>	Prepares the pixel layouts for evaluation.

Public Members

The `KcsXformSeq` class has the following public members.

Table 9-2 `KcsXformSeq` Public Members

Public Member	Description
<pre>virtual KcsStatus addAsParent(KcsXformSeq *parent);</pre>	Adds a parent to my list of parents.
<pre>virtual KcsStatus delAsParent(KcsXformSeq *parent);</pre>	Deletes a parent from my list of parents. Decrements the parent use count and unshares the parent.
<pre>virtual KcsStatus deOptimize();</pre>	Deoptimizes this sequence. If the original data is not available, returns <code>KCS_CANNOT_DEOPTIMIZE</code> .
<pre>virtual KcsStatus evalSegment(const KcsOperationType opsAndHints, KcsPixelFormat *inBuffer, KcsPixelFormat *outBuffer, KcsCallbackFunction progress);</pre>	Evaluates the <code>inBuffer</code> data through the sequence of Xforms described by this instance into <code>outBuffer</code> .
<pre>virtual KcsStatus getLeftmostXform(long *n, KcsXformSeq **pNextXform, KcsXform **nextXform);</pre>	Gets the leftmost basic Xform in the sequence. Returns the parent of the xform(<code>pNextXform</code>) and the number of the Xform within that sequence. Recurses to the true leftmost.

Table 9-2 KcsXformSeq Public Members (Continued)

Public Member	Description
<pre>virtual KcsStatus getNextXform(KcsXformSeq **pNextXform, KcsXform **nextXform);</pre>	Gets the next basic Xform in the sequence. Returns the parent of the xform(<i>pNextXform</i>). Unrolls a sequence into a list of its most basic Xforms. Returns <code>KCS_END_OF_XFORMS</code> when it reaches the last Xform in the sequence.
<pre>virtual long getOrigNumber() const {return(numOriginalXforms);};</pre>	Returns the number of <code>KcsXforms</code> in the sequence.
<pre>virtual KcsStatus getRightmostXform(long *n, KcsXformSeq **pNextXform, KcsXform **nextXform);</pre>	Gets the rightmost basic Xform in the sequence. Returns the parent of the xform(<i>pNextXform</i>) and the number of the Xform within that sequence. Recurses to the true rightmost.
<pre>virtual KcsStatus getXform(long n, KcsXform **anXform);</pre>	Gets the <i>n</i> th Xform in the list. Applies only to this sequence. It does not recurse through sequences of sequences.
<pre>virtual KcsStatus insertXform(long n, KcsXform *anXform);</pre>	Adds the Xform in the <i>n</i> th position in the list. It is not recursive.
<pre>virtual KcsOptimizationType isOptimized();</pre>	Indicates if the sequence is optimized.
<pre>KcsXformSeq(KcsStatus *status, KcsAttributeSet *aAttrSet = NULL);</pre>	Constructors from a sequence of 0 Xforms.
<pre>KcsXformSeq(KcsStatus *status, const KcsOperationType opsAndHints, KcsChunkSet *aChunkSet, KcsChunkId chunkId, KcsAttributeSet *aAttrSet = NULL);</pre>	Constructors from a chunk set object.
<pre>KcsXformSeq(const KcsOperationType opsAndHints, const long numXforms, KcsXform **technologies, KcsCallbackFunction progress, long *numberThatFailed, KcsStatus *stat, KcsAttributeSet *aAttrSet = NULL);</pre>	Constructors based on the list of Xforms supplied.
<pre>virtual ~KcsXformSeq();</pre>	Destructor.
<pre>virtual KcsOptimizationType isOptimized();</pre>	Queries whether this sequence has been optimized.

Table 9-2 KcsXformSeq Public Members (Continued)

Public Member	Description
<pre>virtual KcsStatus optimizeLineage(const KcsOperationType opsAndHints, const KcsOptimizationType optimization, KcsCallbackFunction prog);</pre>	<p>Optimizes this Xform and all of its lineages. Does not optimize parents since a parent use count is kept.</p>
<pre>virtual KcsStatus removeXform(long n);</pre>	<p>Deletes the nth Xform in the list. It is not recursive.</p>
<pre>virtual KcsStatus replaceXform(long n, KcsXform *anXform);</pre>	<p>Replaces the Xform in the nth position in the list with an Xform. It is not recursive.</p>

KcsStatus *Class*

10 

This chapter describes the KCMS framework `KcsStatus` class. This class allows a representation of error and warning values, error text descriptions, error conversions to and from `KcsStatusId`, error comparisons, and external mappings through the C API.

To extend this class write your own version of `findErrDesc` and `findWarningDesc`. See the *KCMS CMM Developer's Guide* for more information on extending this class.

As you read this chapter, you will find it helpful to have access to the `kcsstat.h` header file.

This class does not have any protected members; the public members are described.

Public Members

The `KcsStatus` class has the following public members.

Table 10-1 `KcsStatus` Class Public Methods

Public Member	Description
<code>void deleteDescription();</code>	Deletes the string that describes the status. Use this instead of <code>delete(myDescription)</code> .
<code>const long getCause() const;</code>	Gets the OS or internal cause of the status.
<code>const char *getDescription() const;</code>	Returns a string valid only within the scope of the object; therefore, you must copy it to a separate location.
<code>const KcsStatusId getId() const;</code>	Returns longs, ints, and <code>KcsStatusIds</code> . Set and gets the originating OS- or platform-dependent causes, which are OS- and platform-related as well as device specific. A platform-specific application can map specific errors to native error conditions.
<code>const long getIdLong() const;</code>	Returns a long that corresponds to this <code>KcsStatus</code> .
<code>getlastStatus();</code>	Returns to the last non- <code>KCS_SUCCESS</code> status object.
<code>const long getOwnerId() const;</code>	Returns the owner of the error message.
<code>KcsStatus(const KcsStatus&);</code>	Constructor.
<code>KcsStatus(const KcsStatusId thisId);</code>	Constructor.
<code>KcsStatus(const KcsStausId anId, const char *aDescription);</code>	Constructor.
<code>KcsStatus(const KcsStatusId thisId, const long aCause);</code>	Constructor.
<code>KcsStaus(const int &thisId);</code>	Constructor.
<code>KcsStatus(const int &thisId, const char *myDescription);</code>	Constructor.
<code>KcsStatus(const long &thisId);</code>	Constructor.
<code>KcsStatus(const long &thisId, const char *aDescription);</code>	Constructor.
<code>KcsStatus& operator=(const KcsStatus&);</code>	Copy constructor.
<code>KcsStatus& operator=(KcsStatusId);</code>	Copy constructor.
<code>KcsStatus map();</code>	Takes an internal toolkit error and converts it to an API error.

Table 10-1 KcsStatus Class Public Methods (Continued)

Public Member	Description
<code>void setCause(const long);</code>	Sets the OS or internal cause of the status.
<code>void setDescription(const char * description);</code>	Sets the string that describes the status.
<code>void setId(const KcsStatus anId);</code>	Resets the identifier from an <code>int</code> , <code>long</code> , <code>KcsStatusId</code> , or another <code>KcsStatus</code> object. For example, when a method finds an error, it uses this method to indicate the actual error condition or it can set the identifier equal to a new error. Since the KCMS framework overrides the equal operator, you can set the object equal to <code>KcsStatusId</code> , and the <code>KcsStatus</code> class does the rest. This is the most efficient way to assign <code>KcsStatus</code> objects to particular identifiers.
<code>void setId(const long &anId);</code>	Sets the Id to a <code>long</code> .
<code>void setId(const long, const long &iOwnerId);</code>	Allows other classes to add error strings.

External Loadable Interface

Use these `KcsStatus` external entry points to load your derivatives at runtime. See the DDK manual *KCMS CMM Developer's Guide* for more information on creating a `KcsStatus` derivative as an extension.

Table 10-2 KcsSolarisFile External Loadable Interface

Extern "C"	Description
<code>char *KcsFindErrDesc(KcsStatusId statId);</code>	Creates an instance of the function connecting the custom error codes with your string descriptions. <code>dgettext()</code> would be run on this for localization.
<code>char *KcsFindWarningDesc(KcsStatusId statId);</code>	Creates an instance of the function connecting the custom warning codes with your string descriptions. <code>dgettext()</code> would be run on this for localization.
<code>extern long KcsDLOpenStatCount;</code>	Holds a counter for the number of times this dynamically loadable module has been loaded.

Index

Symbols

~KcsChunkSet(), 23
~KcsFile(), 13
~KcsIO(), 5
~KcsLoadable(), 29
~KcsMemoryBlock(), 10
~KcsProfile(), 35
~KcsProfileFormat(), 43
~KcsShareable(), 1
~KcsSolarisFile(), 17
~KcsTags(), 50
~KcsXform(), 57
~KcsXformSeq(), 63
~KcsXWindow(), 19

A

absRead(), 5
absWrite(), 5
addAsParent(), 62
areLayoutsCloseEnough(), 55
areLayoutsEqual(), 55
aSysError(), 4
attach(), 2, 28
attribute, xvi

C

changePermanentUseCount(), 28
compose(), 55
connect(), 34, 55
connectSink(), 55
connectSource(), 55
convertXform(), 55
copy(), 48
copyData(), 5
createAttributeSet(), 48
createChunkId(), 22
createEmptyProfile(), 32
createIO(), 5
createProfile(), 34
createProfileFormat(), 42
createXform(), 56

D

delAsParent(), 62
deleteChunk(), 22
deleteChunkSetsUIDMapEntries(), 28
deleteDescription(), 66
deleteXform(), 42
deOptimize(), 62
dettach(), 2, 28

dirtyAttrCache(), 42
dirtyXformCache(), 42

E

eval(), 56
evalPrep(), 62
evalSegment(), 62
evaluate(), 34

G

generateLoadWhat(), 43
generateXformAttributes(), 43
get(), 9
getAttribute(), 34, 48
getAttributeInfo(), 48
getAttrSet(), 56
getAttrSize(), 48
getCause(), 66
getChunkId(), 28
getChunkSet(), 28
getChunkSetLength(), 22
getChunkSetOffset(), 23
getChunkSize(), 23
getCmmId(), 43
getComponentDepth(), 56
getDescription(), 66
getEOF(), 5
getFormat(), 34
getFref(), 13
getGlobalCount(), 2
getICProfSeqDescInfo(), 48
getICTextDescInfo(), 49
getICTextDescSize(), 49
getICUcrBgCounts(), 49
getId(), 66
getIdLong(), 66
getIO(), 17, 23
getLastStatus(), 66
getLeftmostXform(), 62

getLoadOrder(), 57
getNextXform(), 63
getNumComponents(), 57
getNumElements(), 9
getObject(), 43
getObjFromUIDMap(), 28
getOffset(), 5
getOpandCont(), 34
getOpsAndHints(), 57
getOrigNumber(), 63
getOwnerId(), 66
getRightmostXform(), 63
getSaveOrder(), 57
getSaveSize(), 43
getSize(), 9
getTag(), 49
getTheCMMId(), 43
getTheCMMVersion(), 43
getTheProfileFormat(), 43
getTheProfileVersion(), 43
getTopChunkId(), 23
getType(), 5
getUseCount(), 2
getXform(), 34, 63
getXformType(), 57

I

iFormat, 32
initDataMembers(), 32
initEmptyFormat(), 43
insertXform(), 63
iOpAndCont, 32
isEqual(), 5, 23
isLoadable(), 28, 35
isLoaded(), 29
isOptimized(), 63
isReadOnly(), 49
isSupported(), 43

K

- KCMS Calibrator Tool, 19
- kcms_calibrate, 19
- kcms_configure, 19
- kcms_server, 16, 18
- KCS_FUNC_INIT_PTR, 28
- KcsAttrCompose(), 49
- KcsAttributeSet class
 - alias to KcsTags class note, 47
- kcschunk.h header file, 21
- KcsChunkSet class, 21 to 25
 - example, 24
 - protected members, 22
 - public members, 22 to 24
- KcsChunkSet(), 23
- KcsClassify(), 50
- KcsCleanupXfrm(), 59
- KcsCommon(), 50
- KcsConcatenate(), 50
- KcsCreateXfrm(), 59
- KcsDLOpenStatCount, 67
- KcsDLOpenXfrmCount, 59
- KcsFile class, 12 to 14
 - examples, 13 to 15
 - public members, 13
- KcsFile(), 13
- kcsfile.h header file, 12
- KcsFindErrDesc(), 67
- KcsFindWarningDesc(), 67
- KcsId, naming convention, xv
- KcsInitXfrm(), 59
- KcsInOut, 54
- KcsIO class, 3 to 7
 - constructor, 5
 - data members, 4
 - destructor, 5
 - examples, 7
 - member function override rules, 6
 - protected members, 4
 - public members, 5 to 6
- KcsIO(), 5
- kcsio.h header file, 3
- KcsIOPosition, 4
- kcsload.h header file, 27
- KcsLoadable class, 27 to 29
 - public members, 28 to 29
- KcsLoadable(), 29
- KcsLoadSaveSet, 57
- kcsmlk.h header file, 8
- KcsMemboryBlock class
 - enumerations, 8
 - protected members, 9
- KcsMemoryBlock class, 8 to 10
 - examples, 10
- KcsmemoryBlock class
 - public members, 9
- KcsMemoryBlock(), 9
- KcsMemoryKind, 8
- KcsNever(), 50
- kcspfmt.h header file, 81
- kcsprofi.h header file, 31
- KcsProfile class, 31 to 38
 - member function override rules, 36
 - protected members, 32 to 33
 - public members, 34 to 36
- KcsProfile(), 32, 35
- KcsProfileFormat class, 41 to 46
 - external loadable interface, 45
 - member function override rules, 44
 - protected members, 42
 - public members, 42 to ??
- KcsProfileFormat(), 42, 43
- kcsshare.h header file, 1, 3
- KcsShareable class, 1 to 2
 - protected members, 1
 - public members, 2
- KcsShareable(), 2
- KcsSolarisFile class, 16 to 17
 - public members, 17
- KcsSolarisFile(), 17
- kcssqlfi.h header file, 16
- kcsstat.h header file, 65
- KcsStatus class, 65 to 67

external loadable interface, 67
public members, 66 to 67
KcsStatus(), 66
KcsTags class, 47 to 51
KcsAttributeSet class alias not, 47
public members, 48 to 51
KcsTags(), 50
kcstags.h header file, 47
KcsTransformKind, 57
KcsUseCSInRight(), 50
KcsUseLeft(), 50
KcsUseRight(), 51
KcsXform class, 53 to 60
enumerations, 54
external loadable interface, 59
member function override rules, 59
public members, 55 to 58
KcsXform(), 57
kcsxform.h header file, 53
KcsXformSeq class, 61 to 64
protected members, 62
public members, 62 to 64
KcsXformSeq(), 63
kcsxfseq.h header file, 61
kcsxwin.h header file, 18
KcsXWindow class, 18 to 20
examples, 19 to 20
public members, 19
KcsXWindow(), 19
kindOfTransform(), 57

L

load(), 29
loadObjectMap(), 42
loadU(), 57

M

map(), 66

N

numberOfCallbacks(), 57
numElements, 9

O

optimize(), 35, 57 to 58
optimizeLineage(), 64

P

pos, 9
position(), 10
postAttrCompose(), 44
propagateAttributes2Xforms(), 35
put(), 10
putObjIntoUIDMap(), 29

R

readChunk(), 23
readChunkBuffer(), 22
relRead(), 5
relWrite(), 6
removeXform(), 64
replaceData(), 6
replaceXform(), 64
returnCurrentNumberOfAttributes(), 51

S

save(), 29, 44, 51
saveNew(), 44
saveObjectMap(), 42
saveTags(), 51
saveU(), 58
setAttribute(), 35, 51
setAttrSet(), 58
setCallbackInterval(), 58
setCause(), 67
setChunkId(), 29
setChunkSet(), 29

setChunkSetOffset(), 23
setChunkSetPrivateOffset(), 24
setCmmId(), 44
setComponentDepth(), 58
setCursorPos(), 6
setDefaultAttributes(), 58
setDescription(), 67
setEOF(), 6
setFref(), 13
setICTextDesc(), 51
setId(), 67
setIO(), 24
setNumComponents(), 58
setObject(), 44
setOffset(), 6
setOpAndCont(), 35
setOpsAndHints(), 58
setTimeAttribute(), 32
setUID(), 29
setXform(), 36
size, 9

T

tag, xvi
tag, *See Also*attribute
terms, equivalent, xvi

U

unLoad(), 29
unloadWhenMatch(), 44
updateChunkUseCount(), 24
updateMonitorXforms(), 33
updatePrinterXforms(), 33
updateScannerXforms(), 33
updateXforms(), 36

V

validateLayouts(), 58

W

writeChunk(), 24
writeChunkBuffer(), 22

X

xformIsNOP(), 36

