Gray Balance Library

Application Program Interface

Pictographics International Corporation
2216 East 117th Street
Burnsville, MN 55337
USA
952-894-6247

Chapter 1: Technical Overview

Device calibration is a very important aspect of color control on today's digital imagers. Even with the wide spread use of device profiles, calibration can keep the underlying imager in a consistent reference state, allowing the profile to have a longer lifetime.

The Gray Balance Library is based on software designed to calibrate several components of the Cymbolic Sciences family of digital imagers (Fire 1000, LightJet 2000 and LightJet 5000). This document frequently refers to the CSI implementation as an example, but the software has been generalized for use in calibrating other devices.

The two calibration components of the Gray Balance Library are described in the following paragraphs.

Exposure Set

The exposure set is a triplet of numbers, one each for red, green and blue. For the LightJet series of imagers, these numbers control the overall drive of the lasers. As the exposure set numbers are increased, more energy is produced by the lasers. Conversely, smaller numbers produce less energy. The exposure set is used to set the minimum or maximum density, depending on the polarity of the material being exposed.

One-dimensional Look Up Tables

While the exposure set will set the desired minimum or maximum density, the remainder of the grayscale is uncalibrated. The one-dimensional look up tables (LUTs) are used to obtain a controlled distribution of densites along the grayscale. In the CSI product line, these LUTs are embodied into a data structure called a *map*, which contains pre-LUTs, a three-dimensional LUT and post-LUTs. Only the post-LUT portions of the map are used to control the density distribution along the grayscale. In the Fire 1000, this is a set of three tables, one each for red, green and blue. Each table contains 1024 entries, whose range is zero through 1023. In the LightJet models, each table contains 4096 entries, whose range is zero through 4095.

Normally, the LUTs are used to produce a "calibrated" grayscale. In this context, calibrated means two things: balance and tone distribution. Balance is the relationship among the red, green and blue density measurements that constitutes neutrality. This rather broad definition allows for different meanings of neutral. Tone distribution refers to the manner in which gray values are distributed along the scale. Different distributions will cause a picture to reproduce differently, for example brighter, darker, more or less contrast and so on.

CSI specifies their definition of a calibrated grayscale by way of a so called target density file. This text file contains the desired red, green and blue density values for a set of digital gray levels, typically 32. Any remaining unspecified levels are to be reconstructed by smooth interpolation. Due to the differences in spectral absorption of photographic dyes, there is usually a different target density file for each type of photographic material. CSI generally specifies target density files in such a way as to produce an approximately uniform perceptual grayscale (i.e. CIE L*).

Chapter 2: Data Structures

There are a few data structures used in this library that will be discussed here. These are defined in `GrayBalLib.h`.

First, a structure is defined that may hold a three-channel color value (e.g. density value, pixel value or exposure set). The implied order is always red, green and blue.

```
typedef int    GbColorInt[3]; /* RGB for density, pixel value or exposure set */
```

When a `GbColorInt` holds a density value, it is always expressed in integer form, scaled by 1000, with rounding. For example, a density value of 2.3456 would be represented as 2346.

Another data structure used in the library is the `GbTarget`.

```
typedef struct GbTarget {          /* specification of a set of target densities */
     int            nTargetSamples;     /* number of samples in set */
     GbColorInt     *digital;           /* set of 'nTargetSamples' digital values */
     GbColorInt     *density;           /* set of 'nTargetSamples' densities */
     int            modMethod;          /* method to modify target densities */
} GbTarget;
```

This structure is used by the caller to specify desired density values for grayscale colors, that is, colors having equal RGB values. The structure may contain any number of samples greater than or equal to two. The number of samples is indicated by `nTargetSamples`, which also indicates the sizes of arrays `digital` (which contains the RGB values in the range [0, 255]) and `density` (which contains the desired density values for this sample). For any given value, `i`, `digital[i][0]`, `digital[i][1]` and `digital[i][2]` must contain the same value. Furthermore, the array `digital` must include both 0 and 255 and must be monotonically increasing.

The `modMethod` value indicates the way in which the target densities should be transformed to align with the actual minimum and maximum densities. The choices are `TARGET_MOD_METHOD_LINEAR`, which linearly transforms each channel of the target densities to match the actual minimum and maximum densities, `TARGET_MOD_METHOD_CLAMP`, which clamps any out of range target densities to the actual range, or `TARGET_MOD_METHOD_LINEAR_MINMAX`, which subjects all three channels of target densities to a common linear transformation that satisfies the following four conditions:

1) $M_c \geq A_c$ for all $c \in \{R, G, B\}$

2) $M_c = A_c$ for at least one of $c \in \{R, G, B\}$

3) $m_c \leq a_c$ for all $c \in \{R, G, B\}$

4) $m_c = a_c$ for at least one of $c \in \{R, G, B\}$

In these conditions, M (m) refers to the modified target maximum (minimum) density, and A (a) refers to the actual maximum (minimum) density.

The `GbMap` data structure consists of three LUTs, for red, green and blue.

```
typedef struct GbMap {
     Flt       lut[3][256];   /* contents in range [0.0, 1.0] */
} GbMap;
```

Chapter 3: API Specifications

The Gray Balance Library consists of three functions for managing calibrations. Table 1 summarizes these functions. Each function is described in the following pages.

Table 1: Summary of Gray Balance Library functions.

Function	Description
`GbExamineDensities`	Examines a set of density measurements for stray values and also measures density errors.
`GbRefineExposureSet`	Refines the current exposure set.
`GbRefineGrayMap`	Refines the current gray balance map.

GbExamineDensities

SYNTAX `int GbExamineDensities(nMeas, measDens, target, report)`

PARAMETERS

`int nMeas`	Number of measurements in array `measDens`.
`GbColorInt *measDens`	Set of `nMeas` density measurements.
`GbTarget *target`	Pointer to the desired target density values for this material.
`GbDensReport *report`	Pointer to where the report information should be placed.

DESCRIPTION The purpose for this function is two-fold. First, it may be used to verify a set of density measurements. Since the test pattern colors are organized in logical trends, the resulting density measurements should also follow a continuous trend. `ExamineDensities` will intelligently examine the set of `nMeas` density measurements found in `measDens` and look for strays, or outliers.

Another useful calculation that `GbExamineDensities` performs is measuring how close the actual measurements are to the target values provided in target. It will check as much as possible for the given set of measurements, such as deviation from the target minimum density, maximum density and gray balance.

The function examines only data sets from one of the defined test patterns. It determines which test pattern the data set represents by looking at `nMeas`, which must be one of those values found in Table 2.

The results of any call to `GbExamineDensities` are placed into report. Any elements of this structure which cannot be determined will be set to the value -1. For example, when examining the measurements from the Exposure Set test pattern, `nMeas` will be set to 13 and the report will contain valid information only in `dMaxError` or `dMinError`, depending on the polarity of the material. Gray balance errors cannot be determined from this test pattern because it does not include a grayscale.

A more interesting example might be a gray balance test pattern with `nMeas` set to 72. All elements of the report can be computed for this set since a full grayscale is included as well as several trends (allowing the measurement of `badMeasMaxErr` and `badMeasStep`).

RETURN VALUE A status code is returned, as defined in `Picto.h`.

GbRefineExposureSet

SYNTAX `int GbRefineExposureSet(inES, outES, stepSize, targetDens, nMeas, measDens)`

PARAMETERS

`GbColorInt inES`	Nominal exposure set used when imaging the test pattern (Table 3, first patch).
`GbColorInt outES`	Refined exposure set values will be placed here (may be same as `inEs`).
`GbColorInt stepSize`	Step sizes for red, green and blue (ΔR, ΔG, ΔB per Table 3). These must be greater than zero.
`GbColorInt targetDens`	Desired target density.
`int nMeas`	Number of measurements in array `measDens` (must be 13).
`GbColorInt *measDens`	Pointer to an array of `nMeas` measurements from the exposure set test pattern.

DESCRIPTION `GbRefineExposureSet` takes a set of measurements (`measDens`) and uses them to refine the current exposure set values (`inES`) so that they better represent the target density (`targetDens`). The new exposure set values are placed into `outES`. Since the exposure set test pattern does not have fixed color values, both the base exposure values (`inES`) and the deviations (`stepSize`) must be provided to `GbRefineExposureSet`.

COMMENTS When creating the test pattern, do not succumb to the natural urge to "zero in" on the perfect exposure set by using a small step size. Understand that `GbRefineExposureSet` uses the device responses in the neighborhood of the base set to model the trends that occur as the exposure values are varied. In any real world system there exist uncertainties, commonly seen in this application as noise, measurement instrument error, light scatter, film emulsion variations, film processing variations, etc. It is important that you choose a step size that is large enough to cause a density change whose magnitude is such that the noise errors are small by comparison.

RETURN VALUE A status code is returned, as defined in `Picto.h`.

GbRefineGrayMap

SYNTAX `int GbRefineGrayMap(inGMap, outGMap, nMeas, measDens, target)`

PARAMETERS

`GbMap *inGMap`	Pointer to the input gray balance map.
`GbMap *outGMap`	Pointer to the where the refined gray balance map should go (may be the same as `inGMap`).
`int nMeas`	Number of measurements in array `measDens`. This must be one of 8, 16, 24, 64, 72, 208, 344, 1024.
`GbColorInt *measDens`	Pointer to an array of `nMeas` density measurements from the gray balance map test pattern.
`GbTarget *target`	Pointer to the target density specification.

DESCRIPTION `GbRefineGrayMap` compares the density measurements from the test pattern (`measDens`) with the desired target densities (`target`) and refines the input gray balance map (`inGMap`), writing the new map out in `outGMap`.

RETURN VALUE A status code is returned, as defined in `Picto.h`

Chapter 4: Test Patterns

The Gray Balance Library makes use of two different test patterns. Each pattern consists of a number of color patches.

This section documents each test pattern in a manner that assigns an index to each patch, and specifies how the color of each patch is to be derived. What is not covered here is the physical layout of the target. This will depend on the material size constraints, the type of color measurement instrument to be used and other factors. It should be noted however, that for all CSI images (Fire and LightJet series), light scatter is a very significant factor which influences the densities of colors, and an effort should be made to arrange the patches in a manner that distributes the intensities on the page as evenly as possible.

The two test patterns will be described in the following pages and are summarized in Table 2.

Table 2: Summary of Gray Balance Library test patterns.

Test Pattern	Number of Patches
Exposure Set	13
Gray Balance Map	(2, 4, 6, 16, 18, 52, 86, or 256) * 4

Exposure Set

The test pattern used to determine exposure sets consists of 13 patches. These measurements are passed into function `GbRefineExposureSet`. Table 3 shows the exposure values needed when imaging each patch. In this table, the base exposure set (parameter `inES` of `GbRefineExposureSet`) is designated by R, G and B. The steps, or deviations from this base (parameter `stepSize` of `GbRefineExposureSet`) are designated ΔR, ΔG and ΔB.

Table 3: Exposure Set Test Pattern Definition.

Index	Red	Green	Blue
0	R	G	B
1	R	G – ΔG	B – ΔB
2	R	G + ΔG	B – ΔB
3	R	G – ΔG	B + ΔB
4	R	G + ΔG	B + ΔB
5	R – ΔR	G	B – ΔB
6	R + ΔR	G	B – ΔB
7	R – ΔR	G	B + ΔB
8	R + ΔR	G	B + ΔB
9	R – ΔR	G – ΔG	B
10	R + ΔR	G – ΔG	B
11	R – ΔR	G + ΔG	B
12	R + ΔR	G + ΔG	B

Gray Balance Map

This is one example of a test pattern used to perform a gray balance. It consists of 4 step wedges of 18 patches each, for a total of 72 patches, whose RGB values are shown in Table 4. This pattern is used with function `RefineGrayMap`.

Table 4: Gray Balance Test Pattern Definition using 72 patches.

Index	Red	Green	Blue	Index	Red	Green	Blue
0	0	0	0	36	0	15	0
1	15	15	15	37	15	30	15
2	30	30	30	38	30	45	30
3	45	45	45	39	45	60	45
4	60	60	60	40	60	75	60
5	75	75	75	41	75	90	75
6	90	90	90	42	90	105	90
7	105	105	105	43	105	120	105
8	120	120	120	44	120	135	120
9	135	135	135	45	135	150	135
10	150	150	150	46	150	165	150
11	165	165	165	47	165	180	165
12	180	180	180	48	180	195	180
13	195	195	195	49	195	210	195
14	210	210	210	50	210	225	210
15	225	225	225	51	225	240	225
16	240	240	240	52	240	255	240
17	255	255	255	53	255	240	255
18	15	0	0	54	0	0	15
19	30	15	15	55	15	15	30
20	45	30	30	56	30	30	45
21	60	45	45	57	45	45	60
22	75	60	60	58	60	60	75
23	90	75	75	59	75	75	90
24	105	90	90	60	90	90	105
25	120	105	105	61	105	105	120
26	135	120	120	62	120	120	135
27	150	135	135	63	135	135	150
28	165	150	150	64	150	150	165
29	180	165	165	65	165	165	180
30	195	180	180	66	180	180	195
31	210	195	195	67	195	195	210
32	225	210	210	68	210	210	225
33	240	225	225	69	225	225	240
34	255	240	240	70	240	240	255
35	240	255	255	71	255	255	240

Note that test patterns with fewer or more patches are also supported, but they must conform to the same structure as this one. The patches must be separated by an equal integer increment (15 in the Table 4 example), and the pattern must step through first equal RGB (index 0 to 17 in Table 4), then greater R,

with equal G and B (index 18 to 35 in Table 4), followed by greater G, with equal R and B (index 36 to 53 in Table 4), and finally greater B, with equal R and G. All possibilities are listed in Table 5.

Table 5: Gray Balance Test Patterns.

Increment	Patches / StepWedge	Total Number of Patches
1	256	1024
3	86	344
5	52	208
15	18	72
17	16	64
51	6	24
85	4	16
255	2	8

Chapter 5: GrayBalLib.h Listing

```
/*          >>> Pictographics Intl. Corp. Confidential and Proprietary <<<
 *   This work contains valuable  confidential and proprietary information.
 *   Disclosure, use or  reproduction without the  written authorization of
 *   Pictographics Intl. Corp. is prohibited.  This  unpublished work by
 *   Pictographics Intl. Corp. is protected by the laws of the United States
 *   and other countries.  If publication of the work should occur the
 *   following notice shall apply:
 *      "Copyright (c) 1990 - 2003 Pictographics Intl. Corp. All Rights Reserved"
 */
/*
 *   Name:     GrayBalLib.h
 *
 *   Author:   B. J. Lindbloom
 *             Tracy Finks
 *
 *   Purpose:
 *             Header file for GrayBalLib routines.
 *
 *   Notes:    All densities are expressed in integer form, scaled by 1000.
 *             For example:
 *
 *             densityInt = (int)(densityFloat * 1000.0 + 0.5);
 *
 *        Gray Balance test patterns must have step sizes of equal integer increments,
 *        so 2, 4, 6, 16, 18, 52, 86, or 256 patch step wedges are supported.
 */
#ifndef GRAYBALLIB_H_
#define   GRAYBALLIB_H_

/* - - - - -  includes  - - - - - */

#include "Picto.h"

/* - - - - -  defines  - - - - - */

/* possible values for 'modMethod' in 'Target' data structure */
#define   GB_TARGET_MOD_METHOD_LINEAR        0    /* linearly transform targets to
actual media range */
#define   GB_TARGET_MOD_METHOD_CLAMP         1    /* match actual targets, if
possible, else clamp */
#define   GB_TARGET_MOD_METHOD_LINEAR_MINMAX 2    /* linearly transform targets to
media min/max range */

/* - - - - -  typedefs  - - - - - */

typedef   int       GbColorInt[3]; /* RGB for density, pixel value or exposure set */

typedef struct GbTarget {          /* specification of a set of target densities */
     int       nTargetSamples;/* number of samples in set */
     GbColorInt     *digital; /* set of 'nTargetSamples' digital values [0, 255] */
     GbColorInt     *density; /* set of 'nTargetSamples' densities */
     int       modMethod;     /* method to be used to modify target densities */
} GbTarget;

typedef struct GbDensReport {      /* report summarizing density analysis */
     int       dMaxError;     /* error in maximum density */
     int       dMinError;     /* error in minimum density */
     int       grayBalRMS;    /* RMS of gray balance error */
     int       grayBalMaxErr; /* maximum gray balance error */
```

```
     int       grayBalStep;   /* step of gray scale that had maximum error [0, 9] */
     int       badMeasMaxErr; /* largest "irregular appearing" measurement, CMY */
     int       badMeasStep;   /* which step had the above error [0, nMeas - 1], CMY */
} GbDensReport;

typedef struct GbMap {
     Flt       lut[3][256];   /* contents in range [0.0, 1.0] */
} GbMap;


/* - - - - -  function prototypes  - - - - - */

#ifdef __cplusplus
extern "C" {
#endif

CLStat
GbExamineDensities(
     int            nMeas,    /* number of measurements in array measDens */
                              /* (one of 8, 13, 16, 24, 64, 72, 208, 344, 1024) */
     GbColorInt     *measDens,/* pointer to array of nMeas density measurements */
     GbTarget       *target,  /* pointer to target density specification */
     GbDensReport   *report   /* pointer to where report should go */
);

CLStat
GbRefineExposureSet(     /* refine current exposure set */
     GbColorInt     inES,     /* current exposure set */
     GbColorInt     outES,    /* new exposure set returned here (may be same as inES)
*/
     GbColorInt     stepSize, /* step sizes (must be > 0) */
     GbColorInt     targetDens,    /* desired target densities */
     int            nMeas,    /* number of measurements in measDens (13 or 30) */
     GbColorInt     *measDens /* pointer to array of nMeas density measurements */
);

CLStat
GbRefineGrayMap(         /* refine current gray balance map */
     GbMap          *inGMap,  /* pointer to input gray balance map */
     GbMap          *outGMap, /* pointer to output gray balance map */
                              /* (may be same as inGMap) */
     int            nMeas,    /* number of measurements in measDens */
                              /* (4 * steps in test pattern) */
                              /* (must be one of 8, 16, 24, 64, 72, 208, 344, 1024) */
     GbColorInt     *measDens,/* pointer to array of nMeas density measurements*/
     GbTarget  *target        /* pointer to target density specification */
);

#ifdef __cplusplus
}
#endif

#endif    /* #ifndef GRAYBALLIB_H_ */
```

Chapter 6: Picto.h Listing

```
/*          >>> Pictographics Intl. Corp. Confidential and Proprietary <<<
 *      This work contains valuable  confidential and proprietary information.
 *      Disclosure, use or  reproduction without the  written authorization of
 *      Pictographics is prohibited.  This  unpublished work by  Pictographics
 *      is protected by the laws of the United States and other countries.  If
 *      publication of the work should occur the following notice shall apply:
 *          "Copyright (c) 1990 - 1999 Pictographics. All Rights Reserved"
 */
/*
 *   Name:     Picto.h
 *
 *   Author:   B. J. Lindbloom
 *
 *   Purpose:
 *             Constants and typedefs common to Pictographics software.
 */

#ifndef PICTO_H_
#define   PICTO_H_

/* - - - - - - - - - - -  Constants  - - - - - - - - - - - */

#define   CE_OK          0         /* ok status */
#define   CE_ARGERROR    -1        /* bad arguments to a function */
#define   CE_GENERROR    -2        /* general error (nothing else fits) */
#define   CE_MALLOCERROR -3        /* memory allocation failure */
#define   CE_OPENERROR   -4        /* file open error */
#define   CE_READERROR   -5        /* file read error */
#define   CE_WRITEERROR  -6        /* file write error */
#define   CE_SEEKERROR   -7        /* file seek error */
#define   CE_FORMATERROR -8        /* data format error */
#define   CE_UNSUPERROR  -9        /* unsupported feature or format */
#define   CE_ABORT       -10       /* user initiated abort */
#define   CE_TIMEOUTERROR -11      /* time out of some process */

#ifndef   TRUE
#define   TRUE           1         /* true */
#undef    FALSE
#define   FALSE          0         /* false */
#endif

#define   RL_UNKNOWN     0         /* unknown color data type or data format */

/* Image file formats */
#define   RL_TIFF        1         /* TIFF format (8-bits per channel) */
#define   RL_TARGA       2         /* TARGA format */
#define   RL_SCITEXCT    3         /* Scitex CT format */
#define   RL_EPS         4         /* EPS format */
#define   RL_DCS         5         /* DCS format */
#define   RL_PCD         6         /* Kodak Photo CD format */
#define   RL_BMP         7         /* Windows Bitmap format */
#define   RL_CINEON      8         /* Kodak Cineon format */
#define   RL_JPEG        9         /* JPEG format */
#define   RL_TIFF16      10        /* TIFF format (16-bits per channel) */

/* Color types */
#define   RL_RGB         1         /* RGB data */
#define   RL_CMYK        4         /* CMYK data */
#define   RL_GRAY        5         /* grayscale data */
```

```
#define   RL_LAB         7         /* CIELAB color data */
#define   RL_YCC8        8         /* PhotoYcc 8-bit color data */

#define   MAX_NCHAN      4         /* maximum permitted number of channels
                                      per scanline */
#define   RL_DEFAULTRES  300       /* default resolution (pixels per inch) */

#define   CC_INPUT       0         /* referring to ColorCircuit inputs */
#define   CC_OUTPUT      1         /* referring to ColorCircuit outputs */

/* Math constants */
#ifndef MATH_CONSTANTS
#ifndef   M_PI
#define   M_PI      3.141592653589793 /* pi */
#endif

#ifndef   M_LN2
#define   M_LN2     0.693147180559945 /* ln(2) */
#endif

#ifndef   M_E
#define   M_E       2.718281828459045 /* e */
#endif

#ifndef   M_GR
#define   M_GR      1.618033988749895 /* golden ratio */
#endif

#ifndef   M_R2
#define   M_R2      1.414213562373095 /* sqrt(2) */
#endif

#ifndef   M_R3
#define   M_R3      1.732050807568877 /* sqrt(3) */
#endif

#ifndef   M_LOG2
#define   M_LOG2    0.301029995663981 /* log10(2) */
#endif
#endif    /* #ifndef MATH_CONSTANTS */

/* - - - - - - - - - -  Macros  - - - - - - - - - - */

#define   COLOR3COPY(a,b)          (b)[0] = (a)[0];         \
                                   (b)[1] = (a)[1];         \
                                   (b)[2] = (a)[2]

#define   COLOR3MAKE(a,b,c,d)      (a)[0] = (b);            \
                                   (a)[1] = (c);            \
                                   (a)[2] = (d)

#define   COLOR3SET(a,b)           (a)[0] = (a)[1] = (a)[2] = (b)

#define   COLOR4COPY(a,b)          (b)[0] = (a)[0];         \
                                   (b)[1] = (a)[1];         \
                                   (b)[2] = (a)[2];         \
                                   (b)[3] = (a)[3]

#define   COLOR4MAKE(a,b,c,d,e)    (a)[0] = (b);            \
                                   (a)[1] = (c);            \
                     (             a)[2] = (d);            \
                                   (a)[3] = (e)
```

```
#define   COLOR4SET(a,b)      (a)[0] = (a)[1] = (a)[2] = (a)[3] = (b)

#define   CLAMP(v,min,max)    if ((v) < (min))         \
                                (v) = (min);         \
                              else if ((v) > (max))    \
                                (v) = (max)

#define   LERP(a,b,c)         (((b) - (a)) * (c) + (a))

#define   ILERP(a,b,c)        ((b) - (a)) / ((c) - (a))

#define   SUM3(a)             ((a)[0] + (a)[1] + (a)[2])

#define   SUM4(a)             ((a)[0] + (a)[1] + (a)[2] + (a)[3])

#define   MAX2(a,b)           ((a) >= (b) ? (a) : (b))

#define   MIN2(a,b)           ((a) <= (b) ? (a) : (b))

#define   MAX3(a,b,c)         (MAX2((a), MAX2((b), (c))))

#define   MIN3(a,b,c)         (MIN2((a), MIN2((b), (c))))

#define   IS_ZERO(v,eps)      (((v) < (eps)) && ((v) > -(eps)))

#define   NOT_ZERO(v,eps)     (((v) > (eps)) || ((v) < -(eps)))

/* - - - - - - - - - -  Typedefs  - - - - - - - - - - */

#ifndef CLStat
#if defined(WIN32) && defined(DLL)
#define   CLStat         __declspec(dllimport) int
#else
#define   CLStat         int
#endif
#endif

#ifndef UCHAR_
typedef unsigned char    uchar;
#define UCHAR_
#endif

#ifndef USHORT_
typedef unsigned short   ushort;
#define USHORT_
#endif

#ifndef UINT_
typedef unsigned int     uint;
#define UINT_
#endif

#ifndef ULONG_
typedef unsigned long    ulong;
#define ULONG_
#endif

#ifndef BOOLEAN_
typedef   unsigned char  boolean;
#define BOOLEAN_
#endif

#ifndef FLT_
```

```
typedef float       Flt;
#define FLT_
#endif

#ifndef FLT2_
typedef double      Flt2;
#define FLT2_
#endif

#ifndef FLTX_
typedef long double FltX;
#define FLTX_
#endif

#ifndef RFLT_
#ifdef CL_DOUBLE_IS_LONG
typedef long double RFlt;
#else
typedef double      RFlt;
#endif
#define RFLT_
#endif

#ifndef MFLT_
#ifdef CL_DOUBLE_IS_LONG
typedef long double MFlt;
#else
typedef double      MFlt;
#endif
#define MFLT_
#endif

#ifndef RATIONAL_
typedef   ulong          rational[2];
#define RATIONAL_
#endif

#ifndef ColorCircuit_
/* Note: Library users should consider ColorCircuit structures to be read-only! */
typedef struct ColorCircuit {
     int       nIn;      /* number of input  channels */
     int       nOut;     /* number of output channels */
     int       iType;    /* color type of input  */
     int       oType;    /* color type of output */
     int       ccBPC;    /* ColorCircuit internal number of bytes per
                            channel {1, 2} */
     int       usrIBPC;  /* user number of bytes per  input channel {1, 2} */
     int       usrOBPC;  /* user number of bytes per output channel {1, 2} */
     void      *data;    /* pointer to other data */
} ColorCircuit;
#define   ColorCircuit_
#endif

#endif    /* #ifndef PICTO_H_ */
```