

Libdc1394 Library support for IEEE 1394 Cameras

– HOW TO

Rohit Agarwal

rohdimp_24@rediffmail.com

Vikram B

vickys_box@rediffmail.com

Revision History

Revision 1.0.1 15th October, 2005 Revised by: Vikram
Minor bug fixes.

Revision 1.0 10th October, 2005 Revised by: Rohit & Vikram
Initial release.

Contents

1. Introduction.....	3
2. Requirements	3
3. Installation.....	3
4. Concepts and Basic Programs.....	9
4.1 Types of functions	12
4.1.1 Get/Query Functions.....	12
4.1.2 Set Functions	15
4.1.3 Print Functions.....	16
4.1.4 Setup Functions	17
4.1.5 Format7 Functions	19
4.1.6 Release Functions	20
4.2 Coriander: A GUI for the libdc1394.....	21
4.3 Example: How to grab image from the IEEE1394 camera.....	22
4.4 How to get color images: Bayer Pattern Concepts.....	31
4.4.1 Nearest Neighbor Algorithm	35
4.4.2 Example: How to grab color image using IEEE1394 camera	36
4.5 Common Problems Faced While Working With IEEE1394 Cameras	44
5. References.....	47
Appendix A.....	48
1. License	48
2. Disclaimer	48
3. About Authors	49
4. Dedications.....	49

1. Introduction

We felt *how-to* on this topic is required since very little documentation is available for libdc1394 library. Moreover, the IEEE1394 cameras are gaining popularity; which will lead to many people using this library to develop applications for cameras on Linux. This *how-to* is an outcome of our experience working with point grey dragonfly IEEE1394 camera, on Linux.

We have given the overview of the library in a layman approach. So anybody who has some programming experience can easily understand the contents.

We followed the following convention through out this document:

The function names and codes are in `courier new`, while the text is in Arial.

We have used “the library” and libdc1394 interchangeably. Similarly IEEE1394 cameras, cameras, IEEE cameras are intended to mean the same.

2. Requirements

Various libraries and modules required are:

Library: libdc1394-1.1.0

Supporting Library: libraw1394-1.2.0

Supporting Modules: ieee1394, raw1394, ohci1394, video1394

Linux Kernel: 2.6.10 or above

3. Installation

Prerequisite: Make sure that the kernel is compiled with IEEE1394 built in support. This is critical!

One can add support for the IEEE1394 devices by compiling the kernel again. Don't panic here are the steps to compile the kernel

1. Go to the directory where the kernel source code is present. Normally it is in `/usr/src`. We would suggest that one should download a fresh copy of the kernel source code (2.6.10) from www.kernel.org

Thus

`cd /usr/src/kernel-2.6.10` (Kernel-2.6.10 is the name of the source code directory. It can vary from system to system)

2. `make menuconfig`
3. The menu will come up. Select `Device Drivers`
4. `Device Drivers` menu will come up. Select `IEEE1394 (Firewire) Support`
5. Mark the following as `<M>` in the menu that is shown after step 4
 - `OHCI-1394 support`
 - `OHCI-1394 Video Support`
 - `OHCI-1394 DVI/O Support`
 - `RAW IEEE1394 I/O Support`
6. Press `ESC` to come out of the menu
7. Save the configuration

Following steps are required to successfully install the library (`libdc1394`).

As a rule of thumb, the installation has to be done as a “root” user.

1) Installation of `libraw1394-1.2.0`

We had this library installed in our Linux, so we skipped this step. If you do not have this library installed, make sure you install it before moving further. Here are the steps for the installation:

- a) `su root`
- b) `tar -xvzf libraw1394-1.2.0.tar.gz`
- c) `cd libraw1394-1.2.0`
- d) `./configure`
- e) `make`
- f) `make install`

2) Installation of libdc1394-1.1.0

- a) `su root`
- b) `tar -xvzf libdc1394-1.1.0.tar.gz`
- c) `cd libdc1394-1.1.0`
- d) `./configure`
- e) `cd libdc1394`
- f) `make`
- g) `make install`

step (a) has to be done only if you are not root

3) Installing the modules

- a) `modprobe ohci1394`
- b) `modprobe video 1394`
- c) `modprobe ieee1394`
- d) `modprobe raw1394`
- e) `lsmod` to view that all the modules:
ieee1394,raw1394,ohci1394,video1394 are now installed.

Note:

The modules have got some dependencies:

ohci1394 → video1394

ieee1394 → video1394, ohci1394, raw1394

So if installing modules by `insmod`, video1394 and raw1394 have to be installed first, followed by ohci1394 and ieee1394

4) Create proper 1394 devices

- a) `cd /dev`
- b) `mknod raw1394 c 171 0`
- c) `chmod 666 /dev/raw1394`
- d) `mkdir video1394`
- e) `cd video1394`
- f) `mknod 0 c 171 16`

- g) `mknod 1 c 171 17`
- h) `chmod 666 /dev/video1394/*`

Note: Every time the system is restarted, the nodes are destroyed and the modules are removed automatically. So, we wrote a script to automate this installation task at the startup, to undo all the damage done during rebooting!

5) `su user`

6) Check for the `LD_LIBRARY_PATH`. It should contain `/usr/local/lib`

This is necessary to run the applications with the help of shared `libdc1394_cotrol.so.13` object file.

Note: One can also set the path by adding `/usr/local/lib` in `/etc/ld.so.conf`

This completes the installation phase.

In order to make sure if the installation is correct, connect the camera to the IEEE1394 card (assuming that the IEEE1394 card is already installed on the system) and just run `testlibraw`.

```
$ testlibraw
```

It tests for the presence of host card, camera and other related parameters. We have given below a typical output from `testlibraw`:

```
Successfully got handle
current generation number: 17
1 card(s) found
nodes on bus: 2, card name: ohci1394
using first card found: 2 nodes on bus, local ID is 0, IRM is 1

doing transactions with custom tag handler
```

```
trying to send read request to node 0... completed with value
0x23127bac
```

```
trying to send read request to node 1... completed with value
0x60217dac
```

```
using standard tag handler and synchronous calls
```

```
trying to read from node 0... completed with value 0x04477dac
```

```
trying to read from node 1... completed with value 0xd37380ac
```

```
testing FCP monitoring on local node
```

```
got fcp command from node 0 of 8 bytes: 01 23 45 67 89 ab cd ef
```

```
got fcp response from node 0 of 8 bytes: 01 23 45 67 89 ab cd ef
```

```
polling for leftover messages
```

The library libdc1394-1.1.0 comes with some example programs which are very helpful in understanding how the programs are to be written. There is a Makefile available in the examples' folder of the library tar file. You can compile the programs and execute.

```
cd libdc1394-1.1.0/examples
```

```
make
```

You can get programs exclusively written for the point grey cameras at:

<http://www.ptgrey.com/support/kb/data/grabdma.tgz>

Fortunately, we have an open source graphical application developed for the IEEE1394 cameras, known as **Coriander**. Coriander provides nice Graphical User Interface for libdc1394 which helps the user to work with the camera more efficiently. We later discuss how Coriander helped us for debugging. We briefly explain how to install Coriander.

Installation of Coriander

Make sure that you have libdc1394 and Gnome libraries (Coriander uses the *gnome.h*) installed on your Linux.

For more details regarding the requirements, look into the user manual provided by the Coriander:

<http://damien.douxchamps.net/ieee1394/coriander/manual.php>

Download the following tar file from sourceforge.net:

coriander-1.0.1.tar.gz

Now, proceed as shown below:

- a) `su root`
- b) `tar -xvzf coriander-1.0.1.tar.gz`
- c) `cd coriander-1.0.1`
- d) `./configure`
- e) `cd src`
- f) `make`
- g) `make install`

run Coriander as

```
cd coriander-1.0.1/coriander-1.0.1/src
./coriander
```

Fortunately, the installation of the libdc1394 and related modules is simple and does not pose any major problems. Some minor problems you may face are due to mistake of not installing it as root or not changing the permissions of the newly created IEEE1394 devices.

4. Concepts and Basic Programs

At present, the documents that are available for understanding libdc1394 are very few. Among the available documents are a FAQ [1] which talks about general concept of the camera and few forums where people have posted their queries.

We had to read the libdc1394 source code to understand how to grab a frame from the camera. The extreme need of a complete explanation on this topic encouraged us to write this *how-to*.

On the basis of what we understood by exploring the source code of the camera, we are presenting an insight of *the library*.

We encourage you to keep referring following files while reading through the next section:

1. `dc1394_control.h`
2. `dc1394_control.c`
3. `dc1394_capture.c`
4. `dc1394_format7.c`

`dc1394_control.h` is the most important file as it contains the declaration of various functions and data structures. It should be read to understand the features supported by IEEE1394 cameras and the functions which access and control various features of camera.

The dc1394 library works closely with raw1394 since all the functions make use of functions of raw1394 (`raw1394_read()` and `raw1394_write()`) in order to retrieve or modify the values of various parameters of the camera.

This is one of the reasons why raw1394 must be installed before installing libdc1394.

To understand how the functions provided by the library work, we need to understand few technical aspects of the IEEE1394 camera:

1. There can be more than one camera that is connected to the host card at any time, hence one need to uniquely identify the camera node.
2. The camera has some control registers to set camera features.
3. According to IEEE specifications, the camera can grab images in different formats. The image formats are defined by two parameters: format and mode. Five modes are defined
 - a) Format_0 contains low resolution mode up to 640x480
 - b) Format_1 contains medium resolution modes : 800x600 and 1024x768
 - c) Format_2 contains mega pixel modes:1280x960 and 1600x1200
 - d) Format_7 is the scalable image format. With this one can change the image size, color coding and other parameters

It may be possible that a camera does not support some of the modes so, we need to be careful while the setup. You use setup to set various parameters like data speed, image mode, framerate that are necessary to make the camera ready to grab the image. We will shortly discuss setup functions.

4. In order to set or get the values of the parameters/features of the camera the library functions modify or read the register values.
5. The camera can work with or without DMA. Unfortunately, our camera does not support DMA and hence we could not really exploit the functionality of DMA related functions. We limit our discussion to non-DMA functions.

With the above background, we begin our discussion about the various functions, data structures and other miscellaneous features available in the `dc1394_control.h`.

We would like to mention one important aspect of the `dc1394_control.h` file:

It lists out the values that can be set for the data speeds, framerates, camera modes, image formats. All the values are in the form of enumerations to help you write readable code.

For example, enumeration for speed indicates that the data speed can be set only as 100,200, 400,800, 1600 or 3200.

`dc1394_control.h` lists the features(brightness, hue, sharpness, saturation, etc) supported for your camera.

Important Data Structures

The library provides some structures which are useful in storing camera and image related information. They have been declared in the file `dc1394_control.h`

Structure	Use
<code>dc1394_camerainfo</code>	Helps in accessing the information about the camera model, vendor, <code>ccr_offset</code> (used to access the registers)
<code>dc1394_cameracapture</code>	Contains various parameters that are to be manipulated while the camera setup. It also declares a buffer of int type that will be used to store the grabbed frame.
<code>dc1394_misc_info</code>	It contains the information about the iso channel id, iso-channel speed, memory channel number.

dc1394_feature_info	Contains the information about a particular feature(brightness, hue, gamma, white balance etc.) for example, availability of a particular feature, value for that feature etc
---------------------	--

We will be largely concerned about the `dc1394_cameracapture` structure.

4.1 Types of functions

We have categorized the functions provided by the library into 6 types. (The categorization is purely done on our own for the ease of discussion). Each of these functions has been declared in `dc1394_control.h`. We suggest you to keep a copy of the file in hand while going through subsequent section.

1. Get/Query Functions
2. Set Functions
3. Print Functions
4. Setup Functions
5. Format7 Functions
6. Release Functions

4.1.1 Get/Query Functions

These functions are used to get the value of various features of the camera, the information about the iso channel, trigger mode, frame rate, format, and mode.

These functions contain string “get” or “query” in their names

For example,

```
int dc1394_get_gamma(raw1394handle_t handle, node_t node , unsigned int
*gamma )
```

This function gets value of the gamma attribute for camera.

Most of the get functions take minimum of three parameters:

- `raw1394handle_t ,node_t node`: these two together identify the camera
- A pointer to obtain the value for example, `int * gamma`

If we try to trace the flow of the function call, we can understand what actually happens:

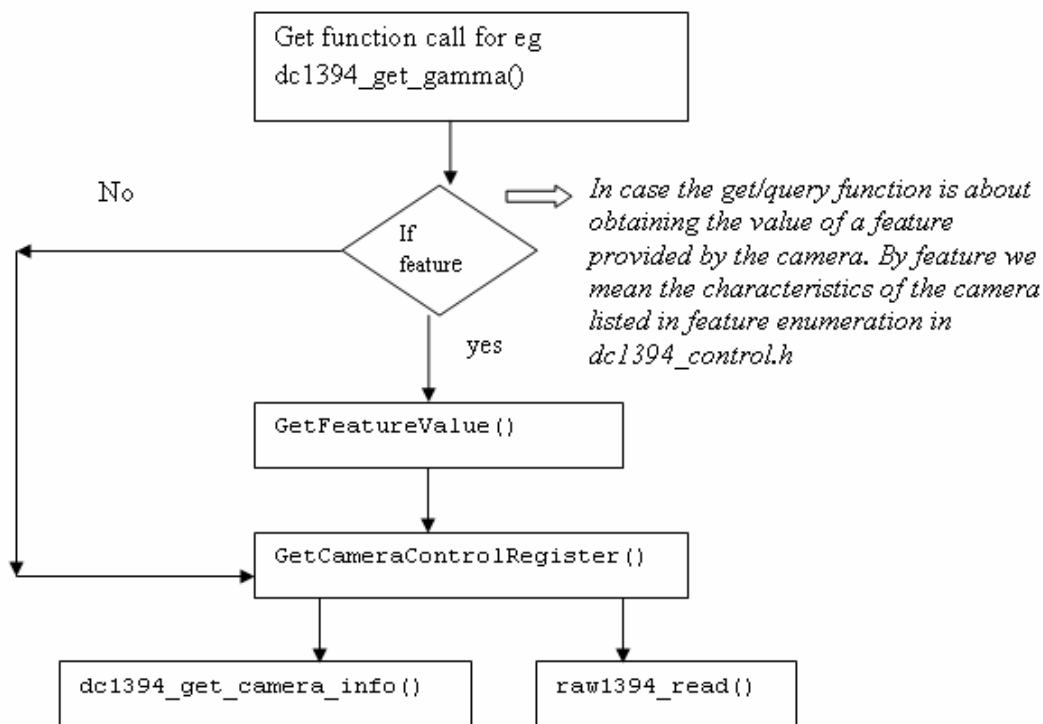


Fig1. Flow of get function call

In case the get function you called is meant to retrieve value of a camera feature (the value for the parameters listed in feature enumeration), that get function calls another get function (`GetFeatureValue`), which takes the enumeration value of the feature as an input. For example, in case of gamma value the function passes 422 as the value (this value can be calculated from the enumeration given in the `dc1394_control.h`)

Ultimately, function `GetCameraControlRegister` whose task is to get the value from the appropriate control registers is called. This function takes the offset value (`octlet_t offset`) which is the offset from the base register as the input to its call. The function prototype is declared in `dc1394_internal.h`

```
GetCameraControlRegister(raw1394handle_t handle, nodeid_t node,
                        octlet_t offset, quadlet_t *value)
```

This will store the final result in the *value*.

Note: The library makes use of typedefed data types `octlet_t`, `quadlet_t` very frequently. These are defined in `dc1394_internal.h` and stand for 8 byte and 4 byte data types.

The `GetCameraControlRegister` will in turn call `dc1394_get_camera_info()` to get the address of the base register:

```
camera->ccr_base = CONFIG_ROM_BASE + info->ccr_offset;
```

Once the base register and the offset is known, `raw1394_read()` is called by `GetCameraControlRegister` to read the actual values.

Now, the `get` function (`dc1394_get_gamma`) use the value returned by the `GetCameraControlRegister` to modify the parameter `gamma`.

```
int dc1394_get_gamma(raw1394handle_t handle, node_t node, unsigned int
*gamma)
```

In this way, the user obtains the value of the camera parameter he queried for.

4.1.2 Set Functions

These functions are used to set the values of various camera features. There is a corresponding set function for almost each get function. You can recognize these functions by searching for “set” string.

For example,

```
dc1394_set_gamma(raw1394handle_t handle, nodeid_t node, int gamma)
```

Like the get functions, this function needs `raw1394handle_t` and `nodeid_t` for camera identification.

The other parameter: `gamma` is the user specified value for the gamma parameter.

The flow of the function call will be quite helpful in understanding what is actually happening. The flow is exactly same as that of the get function .Only difference is this time all the intermediate functions are also set functions and instead of `raw1394_read()` it uses `raw1394_write()` to write the value of the camera parameter on to the registers.

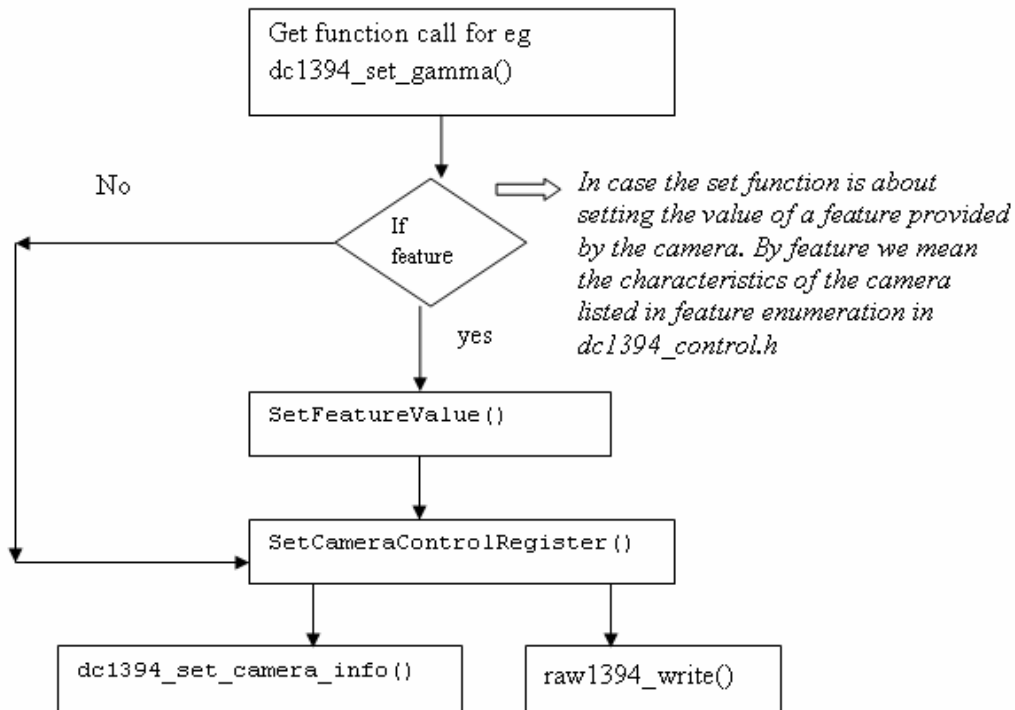


Fig2. Flow of set function call

4.1.3 Print Functions

There are three print functions available:

1. **dc1394_print_camera_info:** This function is used to print the values stored in the `dc1394camera_info` structure. The print function is usually called after `dc1394_get_camera_info()` to ensure that the `dc1394camera_info` structure is not *NULL*.
2. **dc1394_print_feature:** This function is used to print the value of any feature. It takes as an input pointer to `dc1394_feature_info` structure.

3. **dc1394_print_feature_set:** This function is used to print the values of all features present in the camera. It takes as an input pointer to `dc1394_feature_set` structure. The function call basically calls `dc1394_print_feature` repeatedly in a *for* loop.

4.1.4 Setup Functions

As the name suggests these functions are used to prepare the camera to start grabbing the images. In order to setup the camera some parameters are to be passed to the function. The number and the type of parameters are specific to the setup function. But essentially three parameters are to be passed in all the setup functions: `raw1394handle_t`, `nodeid_t` and pointer to `dc1394_cameracapture` structure (this is to provide the buffer for grabbed the image and keep attributes like height of frame, width of frame).

As mentioned previously `raw1394handle_t` and `nodeid_t` uniquely define the camera while pointer to `dc1394_cameracapture` provides the buffer for the image frame to be grabbed. It also stores information about the width and height of the frame which is useful at the time of image processing.

Other parameters that are passed are data speed, frame rate, image format, image mode and iso channel number.

In case of camera using the DMA, number of DMA buffers and drop frames is to be provided. As we have already mentioned, we will not be discussing about the DMA role in this text.

The various setup functions are

1. `dc1394_dma_setup_capture`
2. `dc1394_setup_capture`
3. `dc1394_setup_format7_capture`
4. `dc1394_dma_setup_format7_capture`

These setup functions have been defined in `dc1394_capture.c`.

The user has a choice while passing the values for the parameters like *data-speed, format, mode, and channel*. Instead of providing the value himself he may instruct the function to take the value from the camera. This can be done by passing `QUERY_FROM_CAMERA` in place of the actual value for that function parameter.

The basic flow of control is easy to understand:

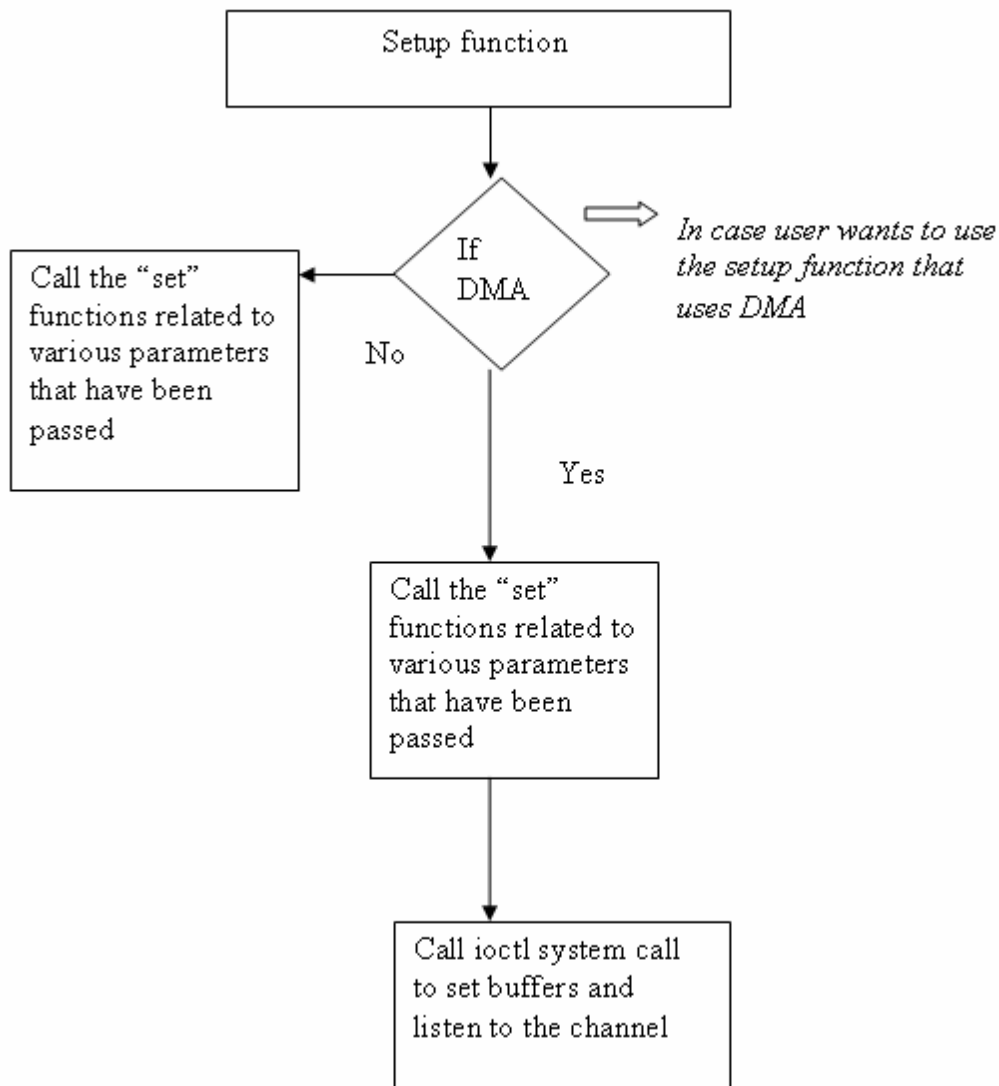


Fig 3. Flow of Setup Function call

As shown in figure 3, the setup function in turn calls various set functions to set the parameter values to the camera registers. In case of DMA setup, after the set functions have been called the ioctl system call is called to set the DMA buffers. The ioctl system calls will fail if the DMA is not supported by the camera and in such case you get the error message:

```
IDEO1394_IOC_LISTEN_CHANNEL ioctl failed
```

The setup functions also allocate memory for the camera _capture buffer

```
camera->capture_buffer= (int*)malloc(camera->quadlets_per_frame*4);
```

4.1.5 Format7 Functions

These functions are used only if the camera is set for format7. This format is preferred since this allows the user to define the size of the image to be captured according to his need. By default the size is 1024x768; you can set it to different dimension, say 960x720.

All format7 functions have “format7” in their function names and the functions have been defined in a separate file `dc1394_format7.c`

The setup function for format7 has a minor difference from the normal setup since it also asks for the size of the frame while you don't have to pass the format parameter as the setup function is meant only for a particular format i.e. format7. The function call flow remains the same as discussed in the previous section

The format7 *get* functions are called *query* functions. The mechanism is different from the normal get/query functions: they don't call `GetCameraControlRegister()` instead, they call `GetCameraFormat7Register()`.

The following flowchart will make the differences evident

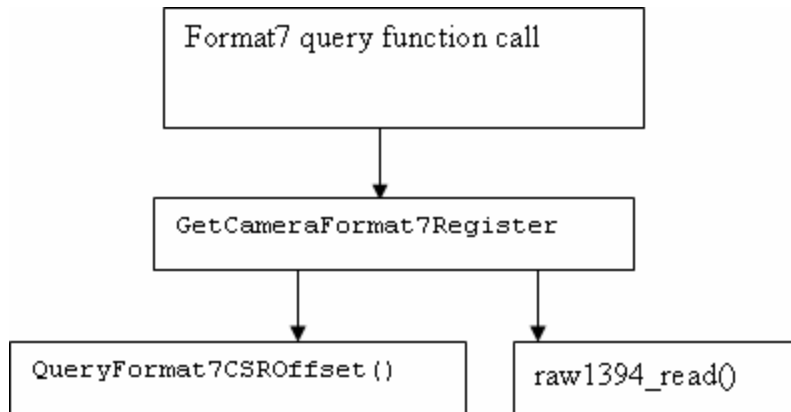


Fig 4. Flow of format7 query function call

The format7 query function will call `GetCameraFormat7Register` which is supposed to read the values from the control and status registers of the camera. This function in turn will call the `QueryFormat` and `CSROffset` to know the offset for the particular information that has been queried. After getting the offset the `raw1394_read` is used to actually read the values.

The format7 set functions also follow the same logic with the obvious difference that the reading functions are replaced by writing functions i.e. `SetCameraFormat7Register` and `raw1394_write()`.

4.1.6 Release Functions

These are the final set of functions identified by us. The basic job of these functions is to release the memory allocated to the capture buffer by the setup routine. This is essential to save the system from memory leak.

These functions are defined in `dc1394_capture.c`:

```
dc1394_release_camera()
```

The function in turn calls `free(camera->capture_buffer)`, which frees the memory.

Similarly the release function is available for DMA setup.

4.2 Coriander: A GUI for the libdc1394

Coriander helps in easy handling of the IEEE1394 cameras. It uses the above discussed functions and libraries and provides a GUI for them. The main advantage of the Coriander is that it saves time that is wasted in camera setup. Also Coriander shows only those features and attributes that are present on the camera and hence you can judge how useful the camera will be for his/her application development. Most important feature of Coriander is its ability to display the captured image at run-time.

Coriander also allows the user to convert a BGGR image to RGB. We will discuss in detail the meaning of these types of images in detail in later sections. Some of the files that can be useful in understanding the functioning of the Coriander are

1. `thread_iso.c`
2. `Camera.c`
3. `main.c`

The Coriander homepage contains an excellent user manual which can be useful in case of any difficulty

<http://damien.douxchamps.net/ieee1394/coriander/manual.php>

Our use of Coriander was limited only to check if the camera was working properly and if the focus was proper. We will give some more usage of the Coriander in later sections.

4.3 Example: How to grab image from the IEEE1394 camera

In this section we will see how to write a small program to grab an image from the camera. We have taken the program (`grab_gray_image.c`) given in the examples in the library tar file. We have removed some lines, to increase the readability of the code. We have provided the explanation for this code below. In order to provide a clear picture of which section of the code does what we have grouped the code lines together which perform a particular task and given a number which corresponds to the explanation bullet number. For example, the code section having number 1 means that user should see explanation point 1 for getting the details.

```
1  #include <stdio.h>
   #include <libraw1394/raw1394.h>
   #include <libdc1394/dc1394_control.h>
   #include <stdlib.h>
   #define IMAGE_FILE_NAME "Image.pgm"

   int main(int argc, char *argv[])
   {
2   FILE* imagefile;
   dc1394_cameracapture camera;
   int numNodes;
   int numCameras;
   raw1394handle_t handle;
   nodeid_t * camera_nodes;

   /* Open ohci and assign handle to it */

3   handle = dc1394_create_handle(0);
   if (handle==NULL)
   {
       fprintf( stderr, "Unable to aquire a raw1394 handle\n\n"
           );
       exit(1);
   }
```

```
}
```

```
/* get the camera nodes and describe them as we find them
```

```
*/
```

```
numNodes = raw1394_get_nodecount(handle);
```

```
camera_nodes = dc1394_get_camera_nodes(handle,&numCameras,1);
```

```
fflush(stdout);
```

```
if (numCameras<1)
```

```
{
```

```
    fprintf( stderr, "no cameras found :(\n");
```

```
    dc1394_destroy_handle(handle);
```

```
    exit(1);
```

```
}
```

```
printf("working with the first camera on the bus\n");
```

```
if( camera_nodes[0] == numNodes-1)
```

```
{
```

```
    fprintf( stderr, "\n"
```

```
        "Sorry, your camera is the highest numbered node\n");
```

```
    dc1394_destroy_handle(handle);
```

```
    dc1394_free_camera_nodes(camera_nodes);
```

```
    exit( 1);
```

```
}
```

```
/*setup capture */
```

```
if (dc1394_setup_capture(handle,camera_nodes[0],
```

```
    0, /* channel */
```

```
    FORMAT_VGA_NONCOMPRESSED,
```

```
    MODE_640x480_MONO,
```

```
    SPEED_400,
```

```
    FRAMERATE_7_5,
```

```
    &camera)!=DC1394_SUCCESS)
```

```
{
```

```
    fprintf( stderr,"unable to setup camera-\n"
```

```
        "check line %d of %s to make sure\n"
```

```
        "that the video mode,framerate and format are\n"
```

```

        "supported by your camera\n",
        __LINE__, __FILE__);
    dc1394_release_camera(handle, &camera);
    dc1394_destroy_handle(handle);
    dc1394_free_camera_nodes(camera_nodes);
    exit(1);
}
dc1394_free_camera_nodes(camera_nodes);

```

7

```

/* set trigger mode */
if( dc1394_set_trigger_mode(handle, camera.node, TRIGGER_MODE_0)
    != DC1394_SUCCESS)
{
    fprintf( stderr, "unable to set camera trigger mode\n");
#ifdef 0
    dc1394_release_camera(handle, &camera);
    dc1394_destroy_handle(handle);
    exit(1);
#endif
}

```

8

```

/* have the camera start sending us data*/

if (dc1394_start_iso_transmission(handle, camera.node)
    !=DC1394_SUCCESS)
{
    fprintf( stderr, "unable to start camera iso transmission\n");
    dc1394_release_camera(handle, &camera);
    dc1394_destroy_handle(handle);
    exit(1);
}

```

9

```

/* capture one frame */

if (dc1394_single_capture(handle, &camera) !=DC1394_SUCCESS)

```



```

{
    fprintf( stderr, "unable to capture a frame\n");
    dc1394_release_camera(handle,&camera);
    dc1394_destroy_handle(handle);
    exit(1);
}

```

```

/* Stop data transmission */

```

10

```

if (dc1394_stop_iso_transmission(handle,camera.node)!=DC1394_SUCCESS)
{
    printf("couldn't stop the camera?\n");
}

```

```

/* save image as 'Image.pgm' */

```

```

imagefile=fopen(IMAGE_FILE_NAME, "w");

```

```

if( imagefile == NULL)
{
    perror( "Can't create '" IMAGE_FILE_NAME "'");
    dc1394_release_camera(handle,&camera);
    dc1394_destroy_handle(handle);
    exit( 1);
}

```

11

```

/* Adding the pgm file header */

```

```

fprintf(imagefile,"P5\n%u %u 255\n", camera.frame_width,
        camera.frame_height );

```

```

/* Writing to the file */

```

12

```

fwrite((const char *)camera.capture_buffer, 1,
        camera.frame_height*camera.frame_width, imagefile);
fclose(imagefile);
printf("wrote: " IMAGE_FILE_NAME "\n");

```

13

```

/* Close camera */
dc1394_release_camera(handle, &camera);
dc1394_destroy_handle(handle);
return 0;
}

```

In order to understand the code properly first understand the general structure of the program properly

1. Include the header files which means include

```

<libraw1394/raw1394.h>
<libdc1394/dc1394_control.h>

```

These are necessary in to access functions of the library

2. Declare three variables of the following data types

```

dc1394_cameracapture
raw1394handle_t
nodeid_t *

```

The `raw1394handle_t` and `nodeid_t *` are required to uniquely identify the camera. Moreover `raw1394handle_t` is used to hold the handle that is created for the OHCI compliant host card.

3. Open ohci and assign a handle to it.

This is done by:

```

handle=dc1394_create_handle(0)

```

Where, `handle` is of `raw1394handle_t` type. The parameter 0 refers to the position of the camera on host card (there may be more than one slot on the host card 0 means that the camera is on the first slot).

In case the wrong number is passed, the handle will not be created.

4. Get camera nodes

There can be more than one camera nodes, since IEEE1394 supports multiple devices on a single port. But, for our discussion we will assume that only a single camera is present. Here is how to get nodes:

```
int numNodes = raw1394_get_nodecount(raw1394handle_t handle)
camera_nodes=dc1394_getcamera_nodes(handle,&numCameras,1)
```

If number of cameras returned is `numCameras <1`, means no camera has been detected.

'1' in the function signifies that a printed description of the camera node found will be shown on the console.

5. Check the condition `camera_nodes[0]==numNodes -1`, it must be false. The explanation for this is out of scope of this document.

6. Call the setup function and pass the various parameters

For example:

Let us examine the setup function of the above example:

```
dc1394_setup_capture(
/* handle and camera_nodes[0] uniquely identifies the camera */
    handle,
    camera_nodes[0],
    0, /* channel */
    /*format of the Image */
    FORMAT_VGA_NONCOMPRESSED,
    MODE_640x480_MONO, /* mode of the image */
    SPEED_400, /* data speed */
    FRAMERATE_7_5, /*Frame rate */
    &camera /*dc1394_cameracapture type pointer */
)!=DC1394_SUCCESS)
```

Our suggestion is that the various parameters should be first queried from the camera and then passed .This is helpful in successful setup since often the user does not know actual values of the various parameters that

have to be passed in the setup function call, and lands up in passing wrong values. In such situations the setup is not done correctly and the camera doesn't get initialized.

We are listing the appropriate get functions that should be called first in order to obtain the correct values of the setup parameters.

- `dc1394_get_iso_channel_and_speed(handle, camera_nodes[0], &channel, &speed) /* to get the channel and the data speed */`
- `dc1394_get_video_format(handle, camera_nodes[0], &format); /*to get the format */`
- `dc1394_get_video_framerate(handle, camera_nodes[0], &framerate); /* to get the framerate*/`
- `dc1394_get_video_mode(handle, camera_nodes[0], &mode); /* to get the mode */`

So the above `dc1394_setup_capture` function call will look like:

```
dc1394_setup_capture(  
/* handle and camera_nodes[0] uniquely identifies the camera */  
    handle,  
    camera_nodes[0],  
    /*we pass the variables instead of actual values */  
    channel ,  
    format,  
    mode,  
    speed,  
    framerate,  
    &camera /*dc1394_cameracapture type pointer */.  
    ) !=DC1394_SUCCESS)
```

7. Setting the trigger mode. This is generally not required. This is just like brightness.

```
dc1394_set_trigger_mode(handle, camera.node, TRIGGER_MODE_0)  
sets the trigger mode to 0.
```

Note: We have passed `camera.node` which indicates that the `dc1394_cameracapture` structure is being used and only the particular node for which the camera has been setup is being referred to.

But we can as well use `camera_nodes[0]`.

8. Have the camera start sending the data to the user. This is done by starting the iso transmission. The following function is used

```
dc1394_start_iso_transmission(handle, camera.node)
```

9. Capture one frame by calling the function

```
dc1394_single_capture(handle, &camera)
```

`camera` is a pointer to the structure `dc1394_cameracapture`. This function will grab the image and store it in the buffer (`capture_buffer`) provided by the structure.

In order to capture more than one frame use a *for* loop and place the function inside it

```
for( i=0; i<100 ;i++) /* to capture 100 images */  
dc1394_single_capture(handle, &camera)
```

10. After the image has been grabbed, stop the data transmission by calling the following function

```
dc1394_stop_iso_transmission(handle, camera.node)
```

11. Add a PGM file header to the captured buffer to see the image using `gimp`.

12. Use `fwrite` to save the captured image, by writing the `buffer(camera.capture_buffer)` in a file. The other parameters like height and width of the image can be extracted from the same structure:

```
camera.frame_width, camera.frame_height.
```

13. Close the camera. This step is necessary to save from the memory leak.

```
dc1394_release_camera(handle,&camera);
dc1394_destroy_handle(handle);
```

In order to compile the program use

```
gcc -o grabImage grabImage.c -ldc1394_control -lraw1394
```

where `grabImage.c` is your program.

We hope that after going through this algorithmic way of explanation you can comfortably understand the example code.

In case we used `format7` image format we have to change only the `setup_capture` function. Let us look at the setup function:

```
dc1394_setup_format7_capture(
/* handle and camera_nodes[0] uniquely identifies the camera */
    handle,
    camera_nodes[0],
    channel, /* channel */
    mode , /*mode */
    bytes_per_packet ,
    left ,/*area of interest start column */
    right, /*area of interest start row */
    width,/* area of interest width */
    height /* area of interest height */
    &camera /* dc1394_cameracapture type pointer */.
    )!=DC1394_SUCCESS)
```

where the values of the parameters `channel`, `speed`, `bytes_per_packet`, `speed`, `mode` are found using the following functions:

- `dc1394_get_iso_channel_and_speed(handle,camera_nodes[0], &channel,&speed) /* to get the channel and the data speed */`
- `dc1394_get_video_mode(handle,camera_nodes[0],&mode); /* to get the mode */`

- `dc1394_query_format7_byte_per_packet(handle, camera_nodes[0], mode, &bytes_per_packet);` /* to get the bytes per packet which depends on the mode*/
- The value of left ,top can be set to `QUERY_FROM_CAMERA` or can be specified directly by the user
- The value of the width and height depends on what is the size of frame the user wants for eg if one wants 960x720 then pass 960 as width and 720 as height.

4.4 How to get color images: Bayer Pattern Concepts

The image grabbed by the sample code in the previous section is not colored (we have intentionally used the word “not colored” since the image is not gray scale either). It is actually Bayer Pattern. We will give an overview of Bayer Pattern and how they are used to get a colored image in this section.

Digital cameras use a solid-state device called an **image sensor**. These fingernail-sized silicon chips contain millions of photosensitive diodes called **photosites**. When you take a picture with a digital camera, the intensity of light hitting each photo site on the sensor is recorded as a signal. Depending on the camera, either 12 or 14 bits of data are recorded. At 12 bits, the camera can record 4,096 levels of brightness. At 14 bits, the camera can record 16,384 levels of brightness. This is referred to as **bit depth**. The higher the bit depth, the finer is the detail, the smoother the transition between tones, and the higher the dynamic range (the ability of the camera to hold detail in both highlight and shadow areas). But at capture, digital images are grayscale, not color. To record color information, each pixel on the sensor is covered with a red, green, or blue filter, with the colors alternating. A common arrangement of color filters is the **Bayer pattern array** that alternates colors, but that also uses twice as many green filters as red and blue. Twice as many green filters are used because our eyes are more sensitive to green. This pattern, or sequence, of filters can vary,

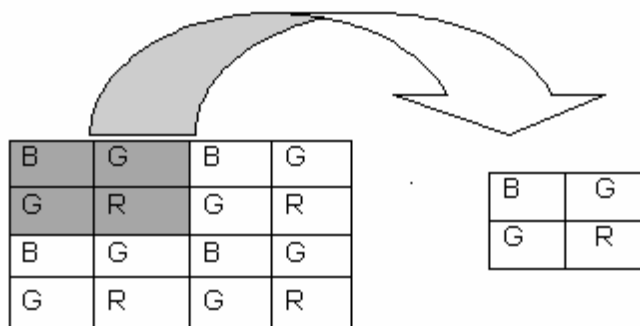
but the widely adopted *Bayer* pattern, which was invented at Kodak, is a repeating 2x2 arrangement. Each pixel has been made sensitive only to one color (one spectral band),

A Typical Bayer Pattern will look like

B	G	B	G
G	R	G	R
B	G	B	G
G	R	G	R

The **tile** or **square (pixel)** that is labeled B means this particular tile is sensitive only to Blue light and so on.

The Bayer patterns may be classified into 4 types depending on how we have arranged the colors. The naming of the Bayer Pattern is done by taking a 2x2 matrix from the top most corner of the pattern and the colors being read in (0,0),(0,1),(1,0),(1,1) order .So for the above Bayer Pattern if we take the 2x2 matrix as



The pattern is therefore known as BGGR

The other possible patterns are:

R	G
G	B

RGGB

G	R
B	G

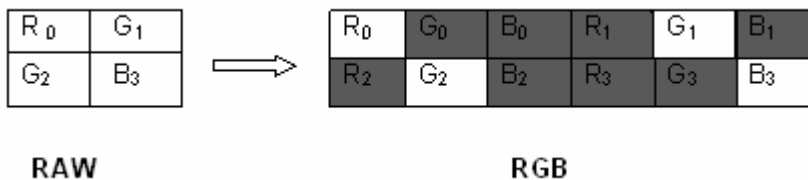
GRBG

G	B
R	G

GBRG

The image we obtained in the previous example was a Bayer pattern Image also known as **RAW** image. This was stored in `camera.capture_buffer`. In order to view what we have captured we convert this RAW image to .PGM by adding a header (look at the explanation point 11 of the previous section).

In order to get a colored image, the Bayer Pattern image is converted to **RGB** image. A RGB image is an enhanced version of Bayer Pattern Image: we try to find the value of the two missing colors at each pixel (Remember that each pixel of the sensor is covered by Bayer Pattern Filter so we get a single color at any pixel by default). This is done by using different algorithms like Nearest Neighbor, Edge Sense etc.



Where, the shaded values are to be calculated by the algorithm. Subscript denotes the tile on the Bayer Pattern to which the Value of R, G, and B belongs. Note that the image size will become 3 times the Bayer Pattern. Now in order to

view the RGB image we convert it to **Bit Map** or **.BMP** image by adding a bitmap header.

To get the clear picture what is happening we have provided the following diagram.

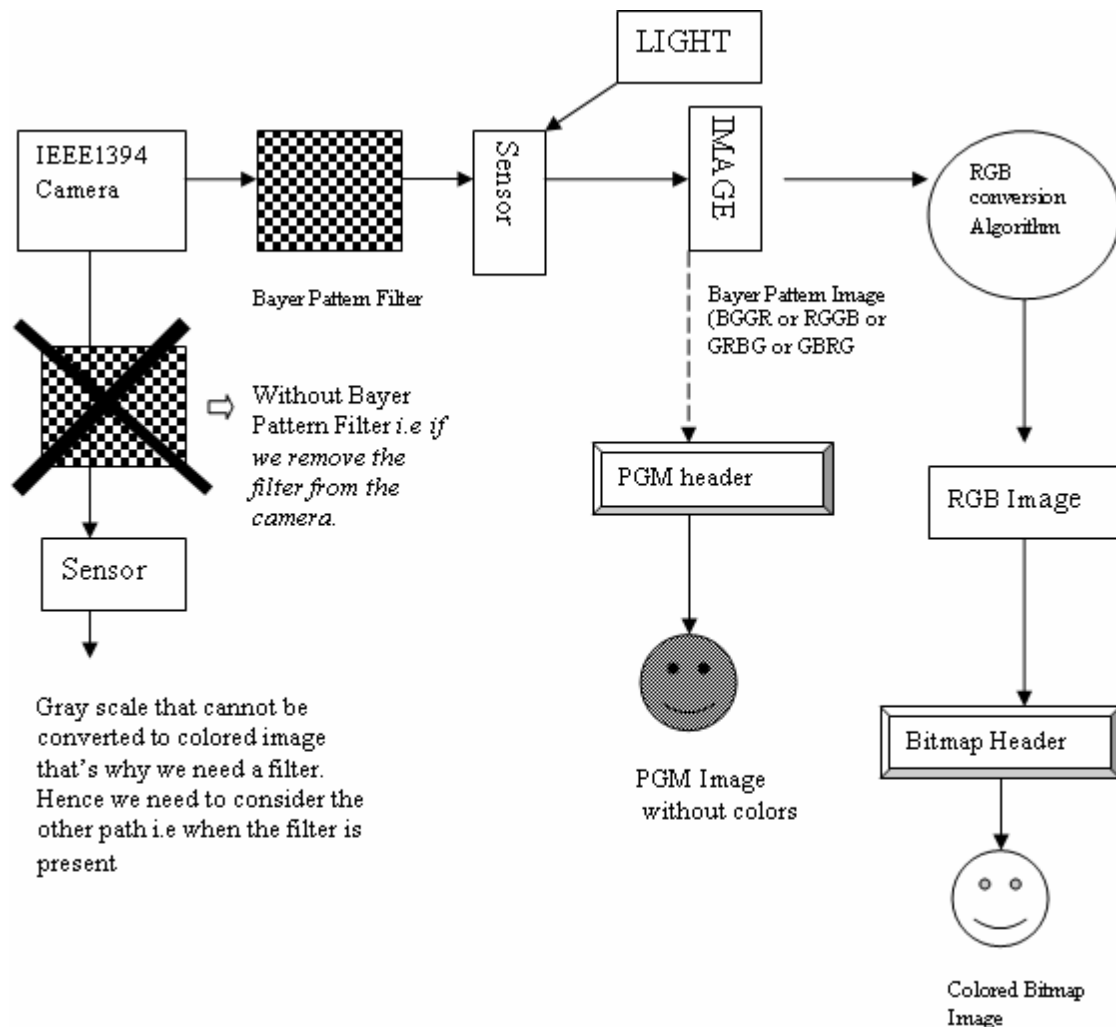


Fig 5. Flow of conversion from Bayer Pattern to Bitmap

Let's understand how the RAW to RGB conversion algorithms work. We will look into detail the Nearest Neighbor algorithm. Other Algorithms are explained in good detail in the following web link

<http://www-ise.stanford.edu/~tingchen/main.htm>

Most these algorithms make use of some kind of interpolations.

4.4.1 Nearest Neighbor Algorithm

In this interpolation method, each interpolated output pixel is assigned the value of the nearest pixel in the input image. The nearest neighbor can be any one of the upper, lower, left or right pixels.

An example will make the logic clear. We try to find the G values for the R, B tiles for a 3x3 block (shown as shaded region. The blank squares either bear R value or B value. We have not them just to make the figure easy to understand). Here we assume the left neighboring pixel value is used to fill the missing ones.

G_1		G_3		G_5
	G_7		G_9	
G_{11}		G_{13}		G_{15}
	G_{17}		G_{19}	
G_{21}		G_{23}		G_{25}

G_7	G_7	G_9
G_{11}	G_{13}	G_{13}
G_{17}	G_{17}	G_{19}

The left hand side table shows the G values for the Bayer Pattern Image. In order to find out the missing G values for the other squares that originally contains only R or B we have the following approach:

Find the nearest G Value square and copy the G value of that square onto the R (B) square .This has been illustrated in the above figure. The square next to G_7 was containing either R or B value. So to get G_8 (G value for square 8) we copied the G value of square 7 since it was the nearest so $G_8 = G_7$. Similarly we filled the other non G value squares.

The same logic is applied when finding R and B values for the Green Squares

4.4.2 Example program to understand how the Colored Image is grabbed using IEEE1394 Camera

Now that we have presented the basic concept of the RAW, RGB and the conversion algorithm we feel that you can understand example program that gives us a colored image.

We have chosen the format as `format7` because the camera we used responds to this format only. We will make use of another example code which provides the implementation of the algorithms. The program is `conversions.cpp` that is available in the `grabdma` folder.

(You can download the `grabdma` folder from:

<http://www.ptgrey.com/support/kb/data/grabdma.tgz>)

To run the code, make sure that you have the following files:

`conversions.h`

`conversions.cpp`

`grabcolor.cpp` has also been taken from the `grabdma` folder. We have modified the code according to our requirements and removed some of the lines. Since most of the code is same as the previously discussed code, we have explained only the portions that are different. These portions have been made **bold**.

```
#include <stdio.h>
#include <libraw1394/raw1394.h>
#include <libdc1394/dc1394_control.h>
#include <stdlib.h>
#include "conversions.h"
```

```
#define IMAGE "Image.rgb"
```

```
int main(int argc, char *argv[])
```

```

{
    FILE* imagefile;
    dc1394_cameracapture camera;
    int numNodes;
    int numCameras;
    raw1394handle_t handle;
    nodeid_t * camera_nodes;
    int channel,speed,mode,bytes_per_packet;

    /* Open ohci and assign handle to it */

    handle = dc1394_create_handle(0);
    if (handle==NULL)
    {
        fprintf( stderr, "Unable to aquire a raw1394 handle\n\n"
                "Please check \n"
                "    - if the kernel modules `ieee1394',`raw1394' and
`ohci1394' are loaded \n"
                "    - if you have read/write access to /dev/raw1394\n\n");
        exit(1);
    }

    /* get the camera nodes and describe them as we find them
    */
    numNodes = raw1394_get_nodecount(handle);
    camera_nodes = dc1394_get_camera_nodes(handle,&numCameras,1);
    fflush(stdout);
    if (numCameras<1)
    {
        fprintf( stderr, "no cameras found :(\n");
        dc1394_destroy_handle(handle);
        exit(1);
    }
    printf("working with the first camera on the bus\n");

    if( camera_nodes[0] == numNodes-1)
    {

```

```

    fprintf( stderr, "\n"
            "Sorry, your camera is the highest numbered node\n");
    dc1394_destroy_handle(handle);
    dc1394_free_camera_nodes(camera_nodes);
    exit( 1);
}

/*obtain the values of the parameter from the camera */
dc1394_get_video_mode(handle,camera_nodes[0],(unsigned int *)&mode);

dc1394_get_iso_channel_and_speed(handle,camera_nodes[0],
(unsigned int *)&channel, (unsigned int *)&speed);

dc1394_query_format7_byte_per_packet(handle,camera_nodes[0],(unsigned)
mode,&bytes_per_packet);


/*setup capture */

if (dc1394_setup_format7_capture(handle,
                                camera_nodes[0],
                                channel , /* channel */
                                mode,
                                speed,
                                bytes_per_packet,
                                0,
                                0,
                                960,
                                720,
                                &camera)!=DC1394_SUCCESS)
{
    fprintf( stderr,"unable to setup camera-\n"
            "check line %d of %s to make sure\n"
            "that the video mode,framerate and format are\n"
            "supported by your camera\n",
            __LINE__,__FILE__);
    dc1394_release_camera(handle,&camera);
}

```

```

    dc1394_destroy_handle(handle);
    dc1394_free_camera_nodes(camera_nodes);
    exit(1);
}
dc1394_free_camera_nodes(camera_nodes);

/* have the camera start sending us data*/

if (dc1394_start_iso_transmission(handle, camera.node)
    !=DC1394_SUCCESS)
{
    fprintf( stderr, "unable to start camera iso transmission\n");
    dc1394_release_camera(handle, &camera);
    dc1394_destroy_handle(handle);
    exit(1);
}

/* capture one frame */

if (dc1394_single_capture(handle, &camera) !=DC1394_SUCCESS)
{
    fprintf( stderr, "unable to capture a frame\n");
    dc1394_release_camera(handle, &camera);
    dc1394_destroy_handle(handle);
    exit(1);
}

/*query the camera to determine the Bayer pattern*/

quadlet_t qValue;
    GetCameraControlRegister( handle,
                              Camera_nodes[0],
                              0x1040,      /* Bayer Tile Mapping register */
                              &qValue );

    bayer_pattern_t pattern = BAYER_PATTERN_BGGR;

```

```

switch( qValue )
{
    case 0x42474752: /* BGGR */
        pattern = BAYER_PATTERN_BGGR;
        break;
    case 0x47524247: /* GRBG */
        pattern = BAYER_PATTERN_GRBG;
        break;
    case 0x52474742: /* RGGB */
        pattern = BAYER_PATTERN_RGGB;
        break;
    case 0x47425247: /* GBRG */
        pattern = BAYER_PATTERN_GBRG;
        break;
    case 0x59595959: /* YYYY = BW */
        fprintf( stderr, "Camera is black and white\n" );
        cleanup();
        return 1;
    default:
        fprintf(stderr, "Camera BAYER_TILE_MAPPING register has an
unexpected value:\n"
"\t0x%x\n", qValue );

        return 1;
}

int bufferSize = camera.frame_width*camera.frame_height;

/*assign a buffer of size three time the original image */
unsigned char* rgbBuffer = new unsigned char[3 * bufferSize];

unsigned char* src = (unsigned char*)camera.capture_buffer;

unsigned char* captureBuffer=
    (unsigned char*)camera.capture_buffer;
for ( int i = 0; i < bufferSize; i++ )
{
    src[i] = captureBuffer[ i * bytesPerPixel ];
}

```



```

    }

    /* convert to color image */

    BayerNearestNeighbor (src,
                          rgbBuffer,
                          camera.frame_width,
                          camera.frame_height,
                          pattern );

    /* Stop data transmission */

    if (dc1394_stop_iso_transmission(handle, camera.node) != DC1394_SUCCESS)
    {
        printf("couldn't stop the camera?\n");
    }

    /* save image as 'Image.rgb' without adding any pgm header */

    printf( "Saving the image...\n" );
    imagefile = fopen( IMAGE, "w" );

    fwrite( rgbBuffer, 3,
            bufferSize, imagefile );

    fclose( imagefile );

    /* Close camera */
    dc1394_release_camera(handle, &camera);
    dc1394_destroy_handle(handle);
    return 0;
}

```

As we have already discussed the use of `GetCameraControlRegister`, you can understand that it has been used to find out the value contained at 0x1040. `libdc1394` does not provide any function to query this address, so we explicitly used this call to get the value.

It is important to understand the utility of the above function call. Refer to our discussion about the Bayer Pattern filters in the previous section. We know that the pattern can be BGGR, RGGB, GRBG, and GRBG. The algorithm that we use for converting the Bayer pattern to RGB requires to know about the type of pattern the camera filter has got so that it can carry out some initialization (refer to `conversions.cpp` for details). The fourth parameter in the function call

```
BayerNearestNeighbor (src,
                      rgbBuffer,
                      camera.frame_width,
                      camera.frame_height,
                      pattern )
```

refers to this value.

`BayerNearestNeighbor` is the function call for the interpolation algorithm we discussed in the last section. This is implemented in `conversions.cpp`.

Having understood this we move on to the **switch-cases**. The value that is obtained for the Bayer Pattern (`qvalue`) is in the hex form which needs to be decoded for setting the value of variable `pattern`.

The case statements indicate the various hex values relating to Bayer Pattern that can be returned by the various cameras. The intimidating values are actually quite easy to decode. Here is the trick:

The hex code for various colors is as follows

42h → B

47h → G

52h → R

Now if the `qvalue` contains 0x42474752 it means: B (42h) G (47h) G (47h) R (52h) or BGGR

Similarly we can decode all the case statements

Finally we need to declare another image buffer that will contain the RGB image. Remember that size of RGB is 3*size of (Bayer Pattern Image)

`rgbbuffer` is therefore assigned 3 times the buffer size, `(camera.capture_buffer)`. This buffer(`rgbbuffer`) will be passed as the destination buffer (the second parameter in `BayerNearestNeighbor`).

After the buffer has been filled with the RGB values we write it in a file `image.rgb`.

In order to view this image using *gimp* we need to append a bitmap header

The method which we employed was:

1. Save the RGB image buffer in a file say `image.rgb`
2. Make a bitmap header and save it in a separate file `bmpheader`
Refer <http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html>
for header details
3. `cat bmpheader image.rgb > bmpfile`
4. open `bmpfile` using gimp

In order to run the program use the following steps:

```
g++ -c conversions.cpp
```

```
g++ -o grabcolor conversions.o grabcolor.cpp -lraw1394 -ldc1394_control
```

Note: There are many algorithms that are available in `conversions.cpp`. So depending upon the requirement you can call the necessary functions.

The **Coriander** is useful in finding out how the RGB image would look like after undergoing different conversion algorithms. For example, it provides the choice between Nearest Neighbor, Edge Sense, Down Sample conversion algorithms. The difference could be observed on a mouse click.

4.5 Common Problems Faced While Working With IEEE1394

Cameras

We now look at some of the common problems faced while working with the camera. We have listed only those problems which we faced while using the camera. These problems are likely to be faced by anyone who uses the library and the camera. Some of the problems were solved with a proper reasoning but some were just hit and trial attempts. We list down the problems and there solutions.

Problem #1

After the installation of the library and the modules the camera did not respond

Sol: We disconnected the camera and connected it again. May be this has to be repeated multiple times. We could not figure out an appropriate reason for this but it works.

Problem #2

Program is not compiling

Sol: If one forgets to link the executable with raw1394 and dc1394_control the program won't compile.

Correct way is

```
gcc -o xx xx.c -lraw1394 -ldc1394_control /*for C files*/
```

```
g++ -o xx xx.cpp -lraw1394 -ldc1394_control /* for Cpp files */
```

Problem #3

Compilation failed giving the error

libdc1394_control.so.13: cannot open the shared object file

Sol: Check for the environment variable LD_LIBRARY_PATH

This should contain the path of the shares library. On our system the path was /usr/local/lib. So the path must be set accordingly.

Problem #4

Program when executed hangs.

Sol: This primarily happens if the values of parameters that are passed to the setup function are not supported by the camera. For e.g. setting the mode `mode_640x480_MONO16` when it is not supported. Our suggestion is that one should always query for the values and then pass them to the function (as explained in the example code in section 4.3).

Problem #5

IDEO1394_IOC_LISTEN_CHANNEL ioctl failed

Error message being displayed

Sol: This means that the DMA support is missing. The error message is a good way to find out if the DMA support is available or not. This can be done by using the setup functions that is meant for DMA support. (We at least found out this way)

Problem #6

Used ctrl c to terminate the image grabbing program. Next time when run the camera program just hangs.

Sol: The error occurs probably because the camera functions like a pipe and experiences a broken pipe situation if terminated without flushing the memory channels.

Trick is to disconnect the camera and connect it again. One can also try debugging the problem installing the `ohci1394`, `video1394`, `raw1394`, `video1394` modules again which works occasionally.

Problem #7

Every parameter is ok and the installation is also fine even then when the program is executed the camera is not detected.

Sol: This is a strange problem which we also face frequently. The solution is based on hit and trial but it always works.

- Disconnect the camera first and then connect it again
- Now run Coriander (that's why we would suggest that one must install **Coriander** before start working with the camera).
- Make all the necessary adjustments i.e. select the mode of the camera, the frame size etc.
- Close Coriander
- Execute your program

Bingo!!! It works.

Note: The first step is critical since we experienced a error message from the Coriander that it could not find the camera if we did not disconnected and connected back the camera before running Coriander.

5. References

1. http://kauri.auck.irl.cri.nz/~johanns/libdc1394/libdc1394_FAQ.html,
libdc1394 FAQ by Johann Schoonees
2. <http://www.ptgrey.com/>
3. <http://www-ise.stanford.edu/~tingchen>
4. <http://damien.douxchamps.net/ieee1394/coriander/manual.php>
5. Point Grey Dragonfly Camera Technical Reference Manual Version
2.1.2.1.3
6. Raw Capture: A Second Chance to Get it Right , Charlotte K. Lowrie,
Sept 1, 2005
7. <http://www.linuxfocus.org>
8. <http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html>

Appendix A

1. License

This document, **Libdc1394 Library support for IEEE 1394 Cameras --How to**, is copyrighted (c) 2005 by Rohit Agarwal & Vikram B. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is available at <http://www.gnu.org/copyleft/fdl.html>.

2. Disclaimer

Although we have tried to do our best to bring out this how to in a good form, we are not responsible for any damage due to the actions taken based upon the information contained in this document. It is impossible to test the things under all the configurations, so probably some of the hints given in this document may be incorrect and may not work on your system. In case you find anything wrong, let us know it first. We will rewrite it as soon as possible. This document is provided ``as is". We put great effort into writing it as accurately as we could, but you use the information contained in it at your own risk. In no event shall we be liable for any damages resulting from the use of this work.

3. About Authors

Rohit Agarwal and Vikram B are pursuing their Masters Degree in IT from International Institute of Information Technology, Bangalore, India.

They can be contacted at

rohdimp_24@rediffmail.com <Rohit>

vickys_box@rediffmail.com <Vikram>

4. Dedications

We are grateful to all our friends for their support. We would specially like to thank Chinmay Narayan our batch mate for his valuable advices. We would like to dedicate this document to our Professor Mr. S.Nagrajan, for he is the person who showed us the path to contribute to the open source community.