Applied Software Engineering
Department for Computer Science
Technische Universität München
Prof. Gudrun Klinker, Ph.D.

Application Programming Interface for Fire-wire Cameras

Dyncon1394

Johannes Schäffner

Betreuer: Dipl.-Inf. Martin Bauer
Aufgabenstellerin: Prof. Gudrun Klinker, Ph.D.

January 2005

Erklärung

Ich versichere, dass ich diese Ausarbeitung des Systementwicklungsprojekts selbstständig verfasst und nur die angegebenen Quellen und Hilfmittel verwendet habe.

München, Januar 2005 Johannes Schäffner

Contents

Chapter 1

Introduction

There are some programs for Linux, which make the use of fire-wire cameras possible.
For example the Coriander Project, which was written by Dan Dennedy, Damien Douxchamps, Yasutoshi Onishi, Xiong Quanren, ctions, Marcus Lim, Andreas Micklei, Olaf Ronneberger and Johan Knol. It provides the possibility to receive pictures from a fire-wire camera and to save them either as an image or as a video, or to do other stuff. Admittedly, this project does not allow the developement of new applications, because it is not an application programming interface but a software to work with.
For that reason it was difficult and expensive to control and to manage fire-wire cameras in Linux. If one wanted to develop a new application, which is using fire-wire cameras, it would require some initial training.
There are two libraries in Linux, which are used, if one wants to develop an application integrating fire-wire cameras. The first one is the LibRAW1394 Library by Andreas Bomb and the second one is the LibDC1394 Library by Gord Peters, Per Dalgas Jakobsen, Chris Urmson, Damien Douxchamps, Dan Dennedy, David Moore, John Stanley and Tim Evers. The LibRAW1394 Library makes the detection and the use of the several cameras possible, which are connected to the fire-wire port, and provides the direct access and manipulation of the cameras' registers (Only cameras built by the IIDC-Specification are supported.).
The LibDC1394 Library is attached to the LibRAW1394 and provides with the help of several structures and functions an easier use of this cameras, so that one does not have to know which registers he has to avail.
Both libraries are written in C, and therefor it may be possible to write C++ programs using this libraries, but it would be much easier and clearer if there was a C++ Interface.
This software is strongly affected by the Coriander Project,because this Project provides the whole functionality, which was wished for. So the first step was to analyse the source code of this Project. However it was difficult to abstract appropriate parts, because of the mergence of the LibDC1394 Library and the Coriander

Code. Thus the Dyncon1394 Library is directly using the LibDC1394 Library and the LibRAW1394 Library and only sporadically some parts from the Coriander Project (e.g. the Conversion Functions or the bus_reset_handler, which has the task to manage plugged in or plugged out cameras). With this software it is now possible to develop applications in shortest time without the need of funded knowledge about the mode of operation of the several fire-wire cameras. One can write programs with only a few lines of code, that are using cameras for example to take pictures and to save them or to display only the captured pictures at the monitor.
Surely there are still some tasks to do, like to decrease the load of the processor during capturing pictures, or to make it easier to upload pictures to a FTP server, et cetera. But the software has still the version number 0.1 and for that reason it is not as powerful as a long term project. Anyhow, it will be easy to expand and to improve the interface, because of the object oriented programming.
The next chapters will dwell on the functionality and the use of the Dyncon1394 api, whereas at chapter 2 all classes and the most important methods are described. You can find code examples at chapter 3, that explain how possible applications could be developed and what you have to heed. Chapter 4 elucidates how extensions and improvements of the api have to be done and chapter 5 explains how to install the software.
In the appendix there are several example programs and tables, that illustrate the use of the software and hopefully facilitate its comprehension.

Chapter 2

Overview

This chapter gives you an overview of all classes and explains what each class stands for. Furthermore it will show you how the different classes worke with each other. At Figure 2.1 you can see in which way the classes are connected and used by each other. At first there is the Dyncon1394Exception Class, that implements

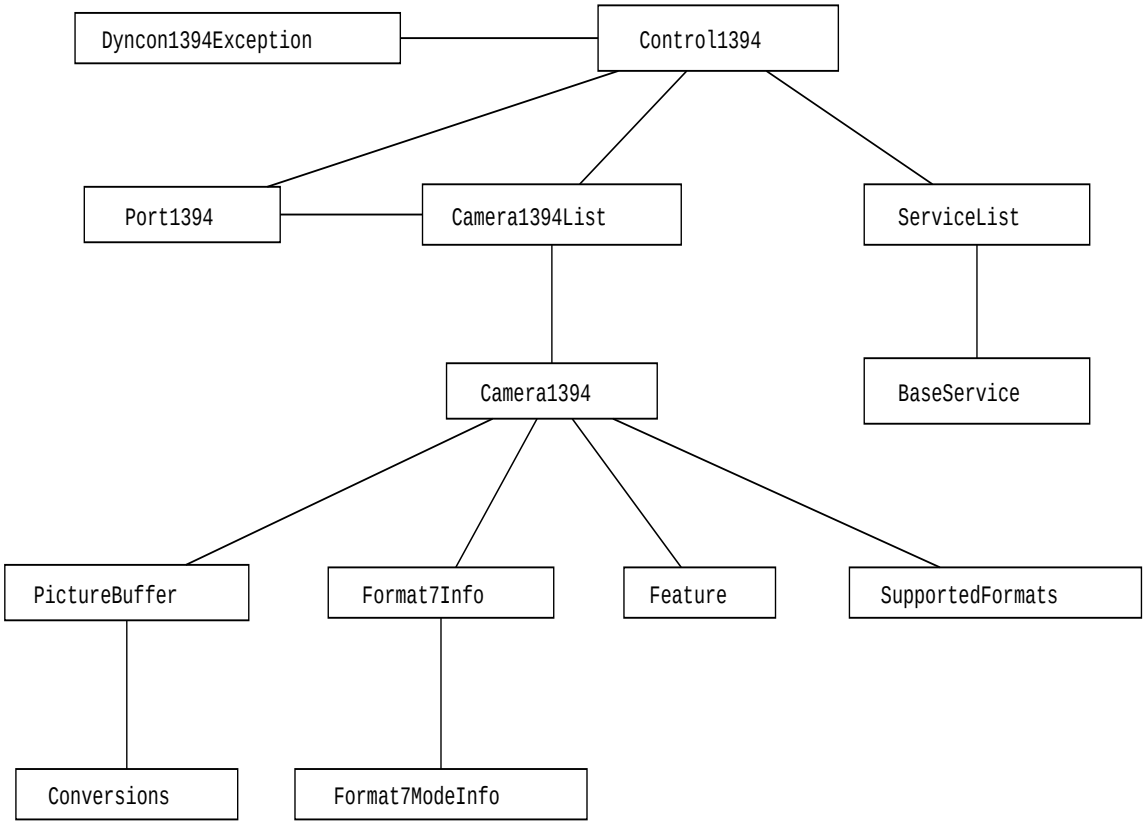

Figure 2.1: Class Diagram

several special exceptions. That way it is possible to catch separately these exceptions, which were thrown by the Dyncon1394 Library. All other classes are using these exceptions.
The Control1394 Class is the main class of the Library. It administrates all cameras, the fire-wire ports and all services. When you develop an application, you only have to create a Control1394 Object, all the rest is done by the constructor of this class.
The Port1394 Class searches for cameras connected to a fire-wire port, sets them up and adds them to the Camera1394List.
The Camera1394List manages all connected cameras within a list.
Every single camera is represented by a Camera1394 Object. Each object holds all necessary informations about one camera, like the node the camera is connected and the handle it is using, but also informations like what formats the camera supports or what features the camera has.
The PictureBuffer Class is used by the Camera1394 Class and stores the captured picture from the camera. It also has the function of converting the picture to the RGB Format or the YUV Format.
The Feature and the SupportedFormats Classes save the informations about the supported formats and the provided features of the camera.
If the camera is able to switch to the format7 mode, the class Format7Info is used.
At last there is the ServiceList Class that manages all services using the captured pictures. This class is necessary because of the bus_reset_handler function, which is called if a camera is plugged out or in. So that the bus_reset_handler has first to stop all displays and then start them again.
In the next sections all important classes, functions and definitions are introduced.

2.1 Definitions

See Appendix B.1 and Appendix B.2 for all informations about the definitions and structures.

2.2 Dyncon1394Exception

There are several different exception for Dyncon1394. All these exceptions are derived from the Dyncon1394Exception Class, which is derived from the exception class of C++. Therefor it is possible to catch either a special Dyncon1394 Exception or to catch all Dyncon1394Exceptions or even to catch all exceptions occurring.

- Dyncon1394Exception
- Dyncon1394_AllocationError

- Dyncon1394_ConvertingError
- Dyncon1394_InitializingError
- Dyncon1394_InvalidArgument
- Dyncon1394_LibdcError
- Dyncon1394_NotSupported
- Dyncon1394_OutOfBounds
- Dyncon1394_RunTimeError

The Dyncon1394Exception is the base-exception for all other exceptions. If you catch this exception all the others will also be caught. The Dyncon1394_AllocationError Exception is thrown when some allocation error has occurred. The Dyncon1394_LibdcError occurred if a call of a LibDC1394 function have returned an error. The function of the other exceptions should be clear because of their name.
If some exception was thrown, one can see the error by calling the what() function each class has implemented.

2.3 Control1394

To use a fire-wire camera connected to the computer, you have to create a Control1394 Object. This class manages all cameras and provides the user the possibility to control them easily. Basically the Control1394 Class consists of a Port1394 Class, a Camera1394List Class and a ServiceList Class. This three classes are controlling the whole system and are controlled by the Control1394 Class.

2.4 Port1394

The Port1394 Class searches for all cameras at the fire-wire bus and adds them to a list managed by the Camera1394List Class. Furthermore, it connects the bus_reset_handler, which will be explained later (see Chapter 2.5).

2.5 Bus Reset Handler

The bus_reset_handler function is called when a reset of the fire-wire bus occurs. This reset is triggered if a new camera was plugged in or another camera was plugged out of the fire-wire card. Then the bus_reset_handler function stops all displays, which are using the captured pictures from the connected cameras, and verifies which camera is the new one or which camera is gone. After done this, the bus_reset_handler function starts all display-functions again.

2.6 Camera1394List

The Camera1394List List is created by the Control1394 Class and filled by the Port1394 Class. It administrates all connected cameras within a list and provides several functions to search for a special camera or to get a camera defined by a number.

2.7 Camera1394

The Camera1394 Class offers all methods to change camera settings, to query the camera status, to request camera features and the supported formats of the camera and to capture pictures from the camera and to convert or save them. Therefor several classes are used.
The SupportedFormats Class stores all available formats and modes for the camera. The Feature Class has the informations about the supported features and the PictureBuffer Class responsible for capturing pictures from camera. Furthermore there is the Format7Info Class, which provides information about all format7 modes and settings.
The class-variables inform about the current camera settings, for example the name of the camera or the node, to which the camera is connected to, and the camera status, for example which format the camera is currently using or which dma device file is to be used for this camera.

2.8 SupportedFormats

When a new Camera1394 Object is created, the LibDC1394 Library offers functions to query all supported formats of the camera. This informations are saved in the SupportedFormats Class and can be queried by the user by several functions, which can be gleaned in [DynAPI].

2.9 Feature

The Constructor of the Camera1394 Class also creates a new Feature Object, which stores informations about all present features and the current status of these features (for example if the camera is able to change the brightness or which value the brightness feature is currently set and if it is set automatically or manually). For this purpose there are methods, like for the supported formats, to change the settings and to query the current status.

2.10 PictureBuffer

If the camera takes a picture, it is saved to a structure provided by the LibDC1394 Library. This structure is embedded in the PictureBuffer Class. In addition to that, it is possible to convert the captured picture either to the RGB Format or to the YUV Format by using conversion methods provided by the Conversions Class.

2.11 Conversions

The methods of the Conversions Class were took over from the Coriander Project. There are several possibilities to convert one picture format to another. See [DynAPI] for more information about that.

2.12 Format7Info

If the camera is able to use the format 7, all information about the different modes and their current settings is saved in this class. The Format7ModeInfo Class stores this informations and the Format7Info Class manages the several Format7ModeInfo Classes (for each mode one Format7ModeInfo Class).

2.13 Format7ModeInfo

Read chapter 2.12 for more information about this class.

2.14 ServiceList

If you want to display the captured pictures on the screen, you have to add your service to the service-list, which is realised by the ServiceList Class. This is necessary, because if a reset occurs, for example if a camera was newly plugged in or out, the bus_reset_handler function has to stop all displaying functions and restart them in the end. If this was not possible, all displaying functions would end by a segmentation fault and the application would abort.

2.15 BaseService

This class is a pure virtual class and has only the two virtual functions start() and stop(). If one wants to develop a displaying program, he has to write a class which is inheriting from the BaseService Class. The stop() function must have the property to shut the displaying function down and the start() function has to be able to restart it.

Chapter 3

Examples of use

This chapter will give you a short introduction on how to use the application programming interface. In the first sections it describes how to start writing a program using a fire-wire camera and how to change the settings of a camera. The following sections cover the question on how to make a camera ready for capturing and how to get a picture from the camera. Finally in the last sections there are some examples on how to work with these pictures and how to use more than one camera in an application.

3.1 How to detect a camera

This section describes what has to be done to start writing an application using a fire-wire camera. First of all you have to include the Dyncon1394 header.

```
#include <control1394.h>
```

Now you have to create a Control1394 Object.

```
Control1394 var_control = new Control1394();
```

The constructor of this class creates a Port1394 Object, a Camera1394List Object and a ServiceList Object. The Constructor of the Port1394 Class searches for all connected cameras on the fire-wire bus and adds them to the camera-list. After this is done, the application programming interface gives the control back to the programmer. Now the system is ready to be used. If any error occurs the application programming interface throws an exception. So it is a good idea to surround all Dyncon1394 calls by a *try ... catch* environment.

3.2 How to change the camera settings

Once the Control1394 Object was created, the system is ready to use the connected cameras. The first step is to look how many cameras are connected. This is done

by the command:

```
int num_cameras = var_control->getCameralist->getSize();
```

After this command one is able to get for example the first connected camera.

```
Camera1394* camera = var_control->getCamera(0);
```

Now one can print the current setting of the camera.

```
camera->printCameraInfo();
camera->printCameraFeatures();
camera->printCameraSupportedFormats();
camera->printCameraMisc();
```

Whereas the first command prints the basic informations about a camera like the model, vendor or the node the camera is connected to, the second command prints all features with the information wether a feature is available or not and what the current settings of a feature are. The third command prints all supported formats and the last command prints the current settings of the camera like the current format, the current mode or for example the current frame rate.
At this point you have all information the camera can deliver and you are now able to change the current settings of the camera. This can be done by the following commands.

```
01 try {
02   camera->setVideoFormat(FORMAT_VGA_NONCOMPRESSED);
03   camera->setVideoMode(MODE_640x480_YUV411);
04   camera->setVideoFramerate(FRAMERATE_15);
05   camera->setIsoChannelAndSpeed(0,SPEED_400)
06   camera->setBrightness(250);
07   camera->setGamma(camera->getGamma()+10);
08   camera->setZoom(40);
09   camera->updateCameraInfo();
10   camera->updateCameraMisc();
11   camera->updateFeatureInfo();
12 }
13 catch(Dyncon1394Exception& e) {
14   cout << e.what() << endl;
15   exit(1);
16 }
```

This are only some examples on how to change the settings of a camera. You can find all commands in the Dyncon1394 API (see [DynAPI]).

All commands are surrounded by the *try ... catch* instruction (it is a recommendable idea to surround the whole code with this instruction). In line 2 the video format of the camera is set to format 0 and in line 3 the video mode is set to 640x480 with YUV411 output. You can refer to all these definitions in the Appendix B.2.
In line 6 the value for the brightness is set to 250, whereas the value for gamma is increased by ten in line 7. After all settings are changed you have to call the update functions, in order to transfer all settings to the Camera1394 Object.
The catch command in the end catches all exceptions that occur during Dyncon1394 calls and print the error message to the standard output.

3.3 How to setup the camera

Once all settings are established, the camera is ready to start capturing. First, you have to setup the camera right. There are two possibilities to do so. The first one is the more difficult one, so it is advisable to use the second option. But for the sake of completeness the first option is also shown.

```
1  try {
2    camera->setup();
3    camera->startIsoTransmission();
4  }
5  catch(Dyncon1394Exception& e) {
6    cout << e.what() << endl;
7    exit(1);
8  }
```

The first command sets the camera up and the second starts the isochronous transmission. This two commands are combined in one command.

```
1  try {
2    camera->startVideo();
3  }
4  catch(Dyncon1394Exception& e) {
5    cout << e.what() << endl;
6    exit(1);
7  }
```

The setup() call uses the settings made in section 3.2.

3.4 How to save a picture

Once the camera is setup, it is ready to receive pictures. For this purpose there are also different options. You can either use one command, which is recommended or

you can use all commands by your own.
All these commands only capture one picture, so if you want to get a video for an application you have to put them into a while-loop or into something similar (see section 3.5 for more informations).

```
1  try {
2    camera->capture();
3    camera->preConvertCapture();
4  }
5  catch(Dyncon1394Exception& e) {
6    cout << e.what() << endl;
7    exit(1);
8  }
```

The capture() call fetches a picture from the camera dependent on whether the camera is using the dma-mode or not. The picture is stored in the picture structure of the PictureBuffer Class.
The preConvertCapture() call copies the picture to the memory area called *image* and does some preconverting like stereo-decoding.
After this two calls the picture is ready for converting to other formats like RGB or YUV.

```
1  try {
2    camera->getVideo();
3  }
4  catch(Dyncon1394Exception& e) {
5    cout << e.what() << endl;
6    exit(1);
7  }
```

The two commands above can be combinded in one single command. So the source code is a little bit easier to read.

```
1  try {
2    camera->getFastVideo();
3  }
4  catch(Dyncon1394Exception& e) {
5    cout << e.what() << endl;
6    exit(1);
7  }
```

This call does in principle the same as the getVideo() call, except that the getFastVideo() call has fewer internal calls and uses the *usleep* command to save some CPU-load. It is up to you which call you use, the result is the same.

```
camera->convertToYUV();
```

After capturing the picture and pre converting it, the picture is prepared to be converted to another format if neccessary.

The first possible format is the YUV 422 format. This call is intended for displaying functions that use the YUV format.

```
camera->convertToRGB24();
```

To convert the picture to the RGB format, you have to use this call. The resulting image is then available in the RGB format with 24 Bits per pixel.

After the image was converted to the RGB format you can save it to disk in various formats. The Dyncon1394 API is using the ImageMagick Library to do so.

```
1  try {
2    camera->savePicture("filename.jpg");
3  }
4  catch(Dyncon1394Exception& e) {
5    cout << e.what() << endl;
6    exit(1);
7  }
```

The type of the file is handed over by the extension of the filename (e.g. if you want to save the image in JPG format, type filename.jpg). If one wants to save the file in the Raw format you have to use this command.

```
1  try {
2    camera->savePicture("filename",1);
3  }
4  catch(Dyncon1394Exception& e) {
5    cout << e.what() << endl;
6    exit(1);
7  }
```

3.5 How to display pictures

If one wants to display the captured pictures to the screen, one basically have to do the same as described above. First you create a Control1394 Object, then you setup the camera and capture a picture. After this you can convert the picture into the required format. Now it is up to you which API you want to use for displaying, but it is important that you write your own class for displaying and let it inherit from the BasicService Class. Then you have to implement the start() and stop() functions. Now you have to register your service to the Servicelist. You can easily do this by the following command.

```
int number_of_service =
         var_control->addService(POINTER_TO_YOUR_CLASS);
```

It is important that you register your service, because if a bus reset occurs, the bus_reset_handler function will shutdown all displaying classes and then it searches for new cameras or it deletes the plugged out camera from the camera list. That is why your start() function has to test if the used camera is still there.
To remove a service from the Servicelist, you have to use the number, that was returned by the addService call.

```
var_control->removeService(number_of_service);
```

Now your service is deregistered.

The last command is to gain access to the converted image.

```
unsigned char* im = NULL;
im = camera->getPictureBuffer()->getImage();
```

At this point your are able to use the image in one of your functions.

3.6 How to use more than one camera

If you want to use more than one camera, it is almost the same as if you use only one camera. The only difference is that you have to select the other cameras as well.

```
01  int count = var_control->getCameralist()->getSize();
02  Camera1394* cameras[size];
03  try {
04    for(int i=0; i < count; i++)
05      cameras[i] = var_control->getCamera(i);
06  }
07  catch(Dyncon1394Exception& e) {
08    cout << e.what() << endl;
09    exit(1);
10  }
```

After this code all available cameras are accessible with the cameras array.

3.7 How to end your program

When your application is finishing you have to do some calls to shut down the camera and to leave the system in a stable state. You do this with the following two calls the second call being optional, for it only turns off the camera.

```
1  try {
2    camera->stopVideo();
3     camera->shutdownCamera();
4  }
5  catch(Dyncon1394Exception& e) {
6    cout << e.what() << endl;
7    exit(1);
8  }
```

In the end you must not forget to delete the *var_control* variable. The destructor of the Control1394 Class destroys then all other classes. You do not have to do more.

```
delete var_control;
```

Now you are ready to use the Dyncon1394 API. For more information about possible functions and that see [DynAPI].

Chapter 4

Extensions and possibilities of improvement

If you want to develop the Dyncon1394 API, this chapter will give you some ideas on how to improve the Dyncon1394 API and how to add some new features to the software.

4.1 How to improve capturing

The main idea to improve capturing is to solve the interdependency between the capturing mechanism and the converting mechanism. One solution to this task is to rewrite the API into several services with the help of threads.
One thread has then the function of capturing the picture from the camera and of pre-converting it. The other threads will then be able to access these pictures and to use them for whatever they are for. For example there could be a thread that converts the picture to the RGB format and saves it to disk, or one that uploads all pictures to a FTP server and so on.
Furthermore, the capturing thread has to use more than only one picture-buffer, because otherwise there will not be any performance advantage compared to the previous solution.
The problem about this possibility is its complexity. The racing condition between the capturing thread and the other threads has to be solved. Otherwise your system will frequently crash.
In some tests it turned out that the load of the CPU was at least halved. The disadvantage of the first tests was that the system was unstable and crashed a lot. But basically it worked and it was only a test version, so it should be possible to improve the performance that way.

4.2 How to add special cameras

If you have a special camera, which is not fully supported, it is imaginable to write a new class, that inherit from the original Camera1394 Class and to add the additional methods there. If there are only some extra features you can change the Features Class as well.
This way it should be possible to support all kinds of cameras, that are available.

4.3 How to add new conversion methods

To add new conversion functions to the Dyncon1394 API, you only have to add them to the Conversions Class or to change some functions there.

Chapter 5

Installation

Requirements

To install the software you will need the library LibRaw1394 as well as the library LibDC1394, whereas the LibDC1394 is an upper part of the LibRAW1394, that provides some easier functions for controlling the fire-wire cameras.
You get both libraries at [sourceforge].
To use Dyncon1394, you should download the newest versions. (Dynon1394 was tested with the version 0.9.0 of LibRAW1394 and the version 1.0.0 of LibDC1394.)

Compiling

After LibRAW1394 and LibDC1394 were be installed, you have to change to the *dyncon1394* directory and there to the *src* directory. Now type *make* and hope that all will go well.

Installation

Once the software has been compiled without an error, you can install the library. For that purpose you have to login as *root*. Now change to the *dyncon1394* directory and copy the file *libdyncon1394.so* from the *lib* directory to the */usr/lib* directory. At last copy the data content of the *include* directory to the */usr/include* directory.
Now the software is ready to be used.

How to use

To use the Dynon1394-API, you have only to include the dyncon1394.h in your own source-code (e.g. `#include <dyncon1394.h>`). At that point it is possible to use application programming interface.

For more advanced informations about how to use the software, see either [DynAPI] or read the first chapters of this article again.

Chapter 6

Summary

Recapitulating, the Dyncon1394 API is stable at this version, but it is surely not developed completely. It is still possible to improve highly the performance of the software and in addition to that you can also add some new services like a simple FTP-service, that upload pictures to a FTP server, or a video-for-Linux-service. Nevertheless, at this state one is able to use this software to develop new applications.

This project was a lot of work and at the same time also a lot of fun, because it made it possible to develop a whole new software, that will hopefully also be of help to other people.
At the beginning there were some difficulties on how to start approaching the task. The Coriander Project, that offers a good possibility to use fire-wire cameras, was therefor the first project I regarded. But it came out that only some parts of this software were useful for this project, as Coriander is a special project, that provides a complete application and it is not suitable for modifying it for other purposes.
Therefor, I have taken over some ideas from this project and some functions like the conversion methods or the bus_reset_handler function.

Finally I hope this application programming interface will be of use for others and I expect it should be developed further.

Appendix A

IIDC-Specifications

This chapter explains how informations from the camera can be read out and how these informations can be used within the software.
For that purpose there will be some tables to make the functionality clear.
This chapter is only for better understanding of the functionality of some parts of the software, but it is not a complete documentation of the IIDC-Specification.
If you want to enhance this software and you want to know more about this Specification, see [IIDC]. There you will find the entire documentation of the IIDC-Specifications.

The next tables are showing the bit-configuration of the register for the video format and of the register for the video mode (format0 only). The last tables are showing the register for the video frame rate (format 0, mode 0 only).

Name	Field	Bit	Description
V_FORMAT_INQ	FORMAT_0	0	VGA non-compressed format
	FORMAT_1	1	Super VGA non-compressed
	FORMAT_2	2	Super VGA non-compressed
	FORMAT_x	3-5	Reserved for other format
	FORMAT_6	6	Still Image Format
	FORMAT_7	7	Partial Image Size Format
	-	8-31	Reserved (All zero)

Table A.1: Bitset for the video format register

0-7	8-15	16-23	24-31
Format	Reserved		

Table A.2: video format register bit configuration

Name	Field	Bit	Description
V_MODE_INQ_0 (Format_0)	Mode_0	0	160x120 YUV444 Mode 24bit/pixel
	Mode_1	1	320x240 YUV422 Mode 16bit/pixel
	Mode_2	2	640x480 YUV411 Mode 12bit/pixel
	Mode_3	3	640x480 YUV422 Mode 16bit/pixel
	Mode_4	4	640x480 RGB Mode 24bit/pixel
	Mode_5	5	640x480 Y (Mono) Mode 8bit/pixel
	Mode_6	6	640x480 Y (Mono16) Mode 16bit/pixel
	Mode_x	7	Reserved for another Mode
	-	8-31	Reserved (All zero)

Table A.3: Bitset for the video mode register

0-7	8-15	16-23	24-31
V_MODE_INQ	Reserved		

Table A.4: video mode register bit configuration

Name	Field	Bit	Description
V_RATE_INQ_0_0 (Format_0, Mode_0)	FrameRate_0	0	Reserved
	FrameRate_1	1	Reserved
	FrameRate_2	2	7.5 fps
	FrameRate_3	3	15 fps
	FrameRate_4	4	30 fps
	FrameRate_x	5-7	Reserved for another frame rate
	-	8-31	Reserved (All zero)

Table A.5: Bitset for the video frame rate register

Appendix B

LibDC1394

B.1 Structures

dc1394_camerainfo

```
1  /* Camera structure */
2  typedef struct __dc1394_camerainfo
3  {
4      raw1394handle_t handle;
5      nodeid_t id;
6      octlet_t ccr_offset;
7      u_int64_t euid_64;
8      char vendor[MAX_CHARS + 1];
9      char model[MAX_CHARS + 1];
10 } dc1394_camerainfo;
```

dc1394_cameracapture

```
1  typedef struct __dc1394_cam_cap_struct
2  {
3      nodeid_t node;
4      int channel;
5      int frame_rate;
6      int frame_width, frame_height;
7      int * capture_buffer;
8      int quadlets_per_frame;
9      int quadlets_per_packet;
10     /* components needed for the DMA based video capture */
11     const unsigned char * dma_ring_buffer;
12     int dma_buffer_size;
13     int dma_frame_size;
14     int num_dma_buffers;
```

```
15      int dma_last_buffer;
16      int num_dma_buffers_behind;
17      const char * dma_device_file;
18      int dma_fd;
19      int port;
20      struct timeval filltime;
21      int drop_frames;
22  } dc1394_cameracapture ;
```

dc1394_miscinfo

```
1   typedef struct __dc1394_misc_info
2   {
3     int format;
4     int mode;
5     int framerate;
6
7     dc1394bool_t is_iso_on;
8     int iso_channel;
9     int iso_speed;
10
11    int mem_channel_number;
12    int save_channel;
13    int load_channel;
14
15  } dc1394_miscinfo;
```

dc1394_feature_set

```
1   typedef struct __dc1394_feature_set_struct
2   {
3     dc1394_feature_info feature[NUM_FEATURES];
4   } dc1394_feature_set;
```

dc1394_feature_info

```
1
2   typedef struct __dc1394_feature_info_struct
3   {
4       unsigned int feature_id;
5       dc1394bool_t available;
6       dc1394bool_t one_push;
7       dc1394bool_t absolute_capable;
8       dc1394bool_t readout_capable;
```

```
9       dc1394bool_t on_off_capable;
10      dc1394bool_t auto_capable;
11      dc1394bool_t manual_capable;
12      dc1394bool_t polarity_capable;
13      dc1394bool_t one_push_active;
14      dc1394bool_t is_on;
15      dc1394bool_t auto_active;
16      char trigger_mode_capable_mask;
17      int trigger_mode;
18      dc1394bool_t trigger_polarity;
19      int min;
20      int max;
21      int value;
22      int BU_value;
23      int RV_value;
24      int target_value;
25
26      dc1394bool_t abs_control;
27      float abs_value;
28      float abs_max;
29      float abs_min;
30
31  } dc1394_feature_info;
```

dc1394bool_t

```
1   /* Yet another boolean data type */
2   typedef enum
3   {
4       DC1394_FALSE= 0,
5       DC1394_TRUE
6   } dc1394bool_t;
```

dc1394videopolicy_t

```
1   /* The video1394 policy: blocking (wait for a frame forever)
2      or polling (returns if no frames in buffer */
3   typedef enum
4   {
5     VIDEO1394_WAIT=0,
6     VIDEO1394_POLL
7   } dc1394videopolicy_t;
```

B.2 Definitions and Enumerations

B.2.1 Formats, Modes and Frame rates

There are several enumerations of the available video-formats and video-modes. It is possible to use the integer numbers as well as the exact name.
These informations can also be found at the file LibDC1394_control.h

Formats

FORMAT_VGA_NONCOMPRESSED	384
FORMAT_SVGA_NONCOMPRESSED_1	385
FORMAT_SVGA_NONCOMPRESSED_2	386
FORMAT_STILL_IMAGE	390
FORMAT_SCALABLE_IMAGE_SIZE	391
FORMAT_MIN	FORMAT_VGA_NONCOMPRESSED
FORMAT_MAX	FORMAT_SCALABLE_IMAGE_SIZE
NUM_FORMATS	FORMAT_MAX - FORMAT_MIN + 1

Figure B.1: Image formats

Format VGA noncompressed - Format 0

MODE_160x120_YUV444	64
MODE_320x240_YUV422	65
MODE_640x480_YUV411	66
MODE_640x480_YUV422	67
MODE_640x480_RGB	68
MODE_640x480_MONO	69
MODE_640x480_MONO16	70
MODE_FORMAT0_MIN	MODE_160x120_YUV444
MODE_FORMAT0_MAX	MODE_640x480_MONO16
NUM_FORMAT0_MODES	MODE_FORMAT0_MAX - MODE_FORMAT0_MIN + 1

Figure B.2: Format 0 modes

Format SVGA noncompressed 1 - Format 1

MODE_800x600_YUV422	96
MODE_800x600_RGB	97
MODE_800x600_MONO	98
MODE_1024x768_YUV422	99
MODE_1024x768_RGB	100
MODE_1024x768_MONO	101
MODE_800x600_MONO16	102
MODE_1024x768_MONO16	103
MODE_FORMAT1_MIN	MODE_800x600_YUV422
MODE_FORMAT1_MAX	MODE_1024x768_MONO16
NUM_FORMAT1_MODES	MODE_FORMAT1_MAX - MODE_FORMAT1_MIN + 1

Figure B.3: Format 1 modes

Format SVGA noncompressed 2 - Format 2

MODE_1280x960_YUV422	128
MODE_1280x960_RGB	129
MODE_1280x960_MONO	130
MODE_1600x1200_YUV422	131
MODE_1600x1200_RGB	132
MODE_1600x1200_MONO	133
MODE_1280x960_MONO16	134
MODE_1600x1200_MONO16	135
MODE_FORMAT2_MIN	MODE_1280x960_YUV422
MODE_FORMAT2_MAX	MODE_1600x1200_MONO16
NUM_FORMAT2_MODES	MODE_FORMAT2_MAX - MODE_FORMAT2_MIN + 1

Figure B.4: Format 2 modes

Format still image - Format 6

MODE_EXIF	256
MODE_FORMAT6_MIN	MODE_EXIF
MODE_FORMAT6_MAX	MODE_EXIF
NUM_FORMAT6_MODES	MODE_FORMAT6_MAX - MODE_FORMAT6_MIN + 1

Figure B.5: Format 6 modes

Format scalable image size - Format 7

MODE_FORMAT7_0	288
MODE_FORMAT7_1	289
MODE_FORMAT7_2	290
MODE_FORMAT7_3	291
MODE_FORMAT7_4	292
MODE_FORMAT7_5	293
MODE_FORMAT7_6	294
MODE_FORMAT7_7	295
MODE_FORMAT7_MIN	MODE_FORMAT7_0
MODE_FORMAT7_MAX	MODE_FORMAT7_7
NUM_MODE_FORMAT7	MODE_FORMAT7_MAX - MODE_FORMAT7_MIN + 1

Figure B.6: Format 7 modes

COLOR_FORMAT7_MONO8	320
COLOR_FORMAT7_YUV411	321
COLOR_FORMAT7_YUV422	322
COLOR_FORMAT7_YUV444	323
MODE_FORMAT7_RGB8	324
MODE_FORMAT7_MONO16	325
MODE_FORMAT7_RGB16	326
COLOR_FORMAT7_MIN	COLOR_FORMAT7_MONO8
COLOR_FORMAT7_MAX	COLOR_FORMAT7_RGB16
NUM_COLOR_FORMAT7	COLOR_FORMAT7_MAX - COLOR_FORMAT7_MIN + 1

Figure B.7: Format 7 color modes

Frame rates

FRAMERATE_1_875	32
FRAMERATE_3_75	33
FRAMERATE_7_5	34
FRAMERATE_15	35
FRAMERATE_30	36
FRAMERATE_60	37
FRAMERATE_MIN	FRAMERATE_1_875
FRAMERATE_MAX	FRAMERATE_60
NUM_FRAMERATES	FRAMERATE_MAX - FRAMERATE_MIN + 1

Figure B.8: Frame rates

B.2.2 Features

Enumeration of features

FEATURE_BRIGHTNESS	416
FEATURE_EXPOSURE	417
FEATURE_SHARPNESS	418
FEATURE_WHITE_BALANCE	419
FEATURE_HUE	420
FEATURE_SATURATION	421
FEATURE_GAMME	422
FEATURE_SHUTTER	423
FEATURE_GAIN	424
FEATURE_IRIS	425
FEATURE_FORCUS	426
FEATURE_TEMPERATURE	427
FEATURE_TRIGGER	428
FEATURE_ZOOM	429
FEATURE_PAN	430
FEATURE_TILT	431
FEATURE_OPTICAL_FILTER	432
FEATURE_CAPTURE_SIZE	433
FEATURE_CAPTURE_QUALITY	434
FEATURE_MIN	FEATURE_BRIGHTNESS
FEATURE_MAX	FEATURE_CAPTURE_QUALITY
NUM_FEATURES	FEATURE_MAX - FEATURE_MIN + 1

Figure B.9: Features

Trigger modes

TRIGGER_MODE_0	352
TRIGGER_MODE_1	353
TRIGGER_MODE_2	354
TRIGGER_MODE_3	355
TRIGGER_MODE_MIN	TRIGGER_MODE_0
TRIGGER_MODE_MAX	TRIGGER_MODE_3
NUM_TRIGGER_MODE	TRIGGER_MODE_MAX - TRIGGER_MODE_MIN + 1

Figure B.10: Trigger modes

B.2.3 Other definitions

Data speed

SPEED_100	0
SPEED_200	1
SPEED_400	2
SPEED_MIN	SPEED_100
SPEED_MAX	SPEED_400
NUM_SPEED	SPEED_MAX - SPEED_MIN + 1

Figure B.11: Data speed

Format7 setup flags

QUERY_FROM_CAMERA	-1
USE_MAX_AVAIL	-2
USE_RECOMMENDED	-3

Figure B.12: Format7 setup flags

Return values

DC1394_SUCCESS	1
DC1394_FAILURE	-1
DC1394_NO_FRAME	-2
DC1394_NO_CAMERA	0xffff

Figure B.13: Return values

Video1394 policy

VIDEO1394_WAIT	0
VIDEO1394_POLL	1

Figure B.14: Video1394 policy

Not classified

MAX_CHARS	32
DC1394_FALSE	0
DC1394_TRUE	1

Figure B.15: Not classified

Appendix C

Example Programs

These example programs can be found in the demo directory of dyncon1394.

Program 1: CamInfo

This program demonstrates how to get informations about a camera.

```
1  #include <iostream>
2  #include <dyncon1394.h>
3
4  using namespace std;
5
6  int main (int argc,char **argv) {
7    cout << "Camera Information" << endl;
8
9    // get control center for cameras
10   try {
11     Control1394* control1394 = new Control1394();
12     // get count of cameras
13     int count = control1394->getCameralist()->getSize();
14     Camera1394* cams[count];
15     // fetch all available cameras
16     for(int i = 0;i< count; i++)
17       cams[i] = control1394->getCamera(i);
18
19     // print information for all cameras
20     for(int i = 0;i < count; i++) {
21       cout << "=========================================================
22       cams[i]->printCameraInfo();
23       cams[i]->printCameraMisc();
24       cams[i]->printCameraFeatures();
25       cams[i]->printSupportedFormats();
```

```
26        cout << "=============================================================
27      }
28      // delete control
29      delete control1394;
30    }
31    catch(Dyncon1394Exception& e) {
32      cout << e.what() << endl;
33    }
34    return 0;
35  }
```

Program 2: SavePic

This program captures a frame from the camera and saves it to disk.

```
1   #include <iostream>
2   #include <dyncon1394.h>
3
4   using namespace std;
5
6   int main (int argc,char **argv) {
7     cout << "Save pciture of camera" << endl;
8
9     try {
10      // get control over cameras
11      Control1394* control1394 = new Control1394();
12
13      int count = control1394->getCameralist()->getSize();
14      if(count < 1) {
15        cout << "There is no camera\n";
16        exit(1);
17      }
18
19      // get first camera
20      Camera1394* cam = control1394->getCamera(0);
21
22      // stop video, if camera is active
23      cam->stopVideo();
24
25      // set video mode to 640x480 YUV 4:1:1
26      cam->setVideoMode(66);
27
28      // iso channel 0, camera speed 2 (400 Mbps)
29      cam->setIsoChannelAndSpeed(0,2);
```

```
30
31      // set to 30 fps
32      cam->setVideoFramerate(36);
33
34      // update settings
35      cam->updateCameraMisc();
36
37      // setup camera
38      cam->startVideo();
39
40      // get picture
41      cam->getVideo();
42
43      // convert picture to rgb24
44      cam->convertToRGB24();
45
46      // resolution of picture
47      cout << "Camera resolution: " << cam->getPictureBuffer()->getWidth(
48           <<  "x" << cam->getPictureBuffer()->getHeight() << endl;
49
50      // stop receiving pictures
51      cam->stopVideo();
52
53      // cam save picture
54      cam->savePicture("picture.jpg",1);
55
56      // delete control
57      delete control1394;
58    }
59    catch(Dyncon1394Exception& e) {
60      cout << e.what() << endl;
61    }
62    return 0;
63  }
```

Program 3: Settings

This program demonstrates how to change the camera settings

```
1   #include <iostream>
2   #include <dyncon1394.h>
3
4   using namespace std;
5
```

```
6  int main (int argc,char **argv) {
7    cout << "Use more cameras" << endl;
8
9    try {
10     // get control over cameras
11     Control1394* control1394 = new Control1394();
12
13     // get count of cameras
14     int count = control1394->getCameralist()->getSize();
15
16     if(count < 1) {
17       cout << "No camera found\n";
18       delete control1394;
19       exit(1);
20     }
21
22     Camera1394* cam;
23
24     // get camera
25     cam = control1394->getCamera(0);
26
27     // stop video in case the camera is active
28     cam->stopVideo();
29
30     // set camera to factory settings
31     cam->setFactorySettings();
32
33     // update current settings
34     cam->updateCameraMisc();
35
36     // update camera information
37     cam->updateCameraInfo();
38
39     // set video mode to 640x480 YUV 4:1:1
40     cam->setVideoMode(66);
41
42     // set Iso channel to 0 and Iso speed to 2 (400Mbps)
43     cam->setIsoChannelAndSpeed(0,2);
44
45     // set framerate to 30 fps
46     cam->setVideoFramerate(36);
47
48     if(cam->isFeaturePresent(FEATURE_HUE)) {
49       cout << "Before any setting - Hue = " << cam->getHue() << endl;
```

```
50
51        // increase hue by 10
52        cam->setHue(cam->getHue()+10);
53
54        cout << "After first setting - Hue = " << cam->getHue() << endl;
55
56        // decrease hue by 10
57        cam->setHue(cam->getHue()-10);
58
59        cout << "After second settig - Hue = " << cam->getHue() << endl;
60      }
61
62      if(cam->isFeaturePresent(FEATURE_BRIGHTNESS)) {
63        cout << "Brightness is set to " << cam->getBrightness() << endl;
64        // get current value for brightness
65        int oldvalue = cam->getBrightness();
66
67        // get maximum for brightness
68        int maxvalue = cam->getFeature(FEATURE_BRIGHTNESS)->getMaxValue(
69        cout << "Maximum value for brightness: " << maxvalue << endl;
70
71        // set brightness to maximum
72        cam->setBrightness(maxvalue);
73
74        cout << "Brightness is set to " << cam->getBrightness() << endl;
75
76        // set brightness to old value
77        cam->setBrightness(oldvalue);
78        cout << "Brightness is set to " << cam->getBrightness() << endl;
79        }
80
81      // delete camera control
82      delete control1394;
83    }
84    catch(Dyncon1394Exception& e) {
85      cout << e.what() << endl;
86    }
87    return 0;
88  }
```

Program 3: UseMoreCams

This program demonstrates the use of more than one camera.

```
1  #include <iostream>
2  #include <dyncon1394.h>
3
4  using namespace std;
5
6  int main (int argc,char **argv) {
7    cout << "Use more cameras" << endl;
8
9    try {
10     // get control over cameras
11     Control1394* control1394 = new Control1394();
12
13     // get count of cameras
14     int count = control1394->getCameralist()->getSize();
15
16     Camera1394* cams[count];
17
18     // fetch cameras
19     for(int i=0;i < count; i++)
20       cams[i] = control1394->getCamera(i);
21
22     // setup cameras
23     for(int i=0; i < count; i++) {
24
25       cams[i]->stopVideo();
26
27       cams[i]->setFactorySettings();
28
29       cams[i]->updateCameraMisc();
30
31       cams[i]->updateCameraInfo();
32
33       cams[i]->setVideoMode(66);
34
35       cams[i]->setIsoChannelAndSpeed(i,2);
36
37       cams[i]->setVideoFramerate(36);
38     }
39
40     // start cameras and fetch one rgb24 picture from each camera
41     for(int i=0;i< count; i++) {
42
43       cams[i]->startVideo();
44
```

```
45        cams[i]->getVideo();
46
47        cams[i]->convertToRGB24();
48      }
49
50      // stop cameras and save pictures
51      for(int i = 0;i < count; i++) {
52
53        cams[i]->stopVideo();
54
55        char* filename = (char*)malloc(sizeof(char)*25);
56        sprintf(filename,"Picture_Camera%i.jpg",i+1);
57
58        cams[i]->savePicture(filename,1);
59      }
60
61      // delete camera control
62      delete control1394;
63
64    }
65    catch(Dyncon1394Exception& e) {
66      cout << e.what() << endl;
67    }
68    return 0;
69  }
```

Program 5: Display

This program sets a camera up and displays the captured frames. It is not registering itself to the Servicelist, so it will crash if the camera is plugged out.

```
1   #include <iostream.h>
2   #include <ptc.h>
3   #include <dyncon1394.h>
4
5   int main()
6   {
7     Control1394* control1394;
8     Camera1394* camera;
9     try {
10      control1394 = new Control1394();
11
12      camera = control1394->getCamera(0);
13
```

```
14      if(camera == NULL) {
15        cout << "No camera found\n";
16        delete control1394;
17        exit(1);
18      }
19
20      camera->printCameraInfo();
21      camera->setVideoMode(66);
22      camera->setIsoChannelAndSpeed(0,2);
23      camera->setVideoFramerate(36);
24      camera->printCameraMisc();
25
26      camera->startVideo();
27
28      unsigned int width = 640;
29      unsigned int height = 480;
30      Console *console = new Console();
31      Format format(24,0xFF0000,0xFF00,0xFF);
32      try {
33        // try to open the console matching the image resolution
34        console->open("Display Camera Picture",width,height,format);
35      }
36      catch (Error&) {
37        // fallback to the default resolution
38        console->open("Display Camera Picture",format);
39      }
40
41      //Surface surface(width,height,format);
42      while (true) {
43        if (console->key()) {
44          // read console key press
45          const Key key = console->read();
46
47          try {
48            switch (key.code()) {
49            case Key::LEFT:
50              camera->setBrightness(camera->getBrightness()+10);
51              break;
52            case Key::RIGHT:
53              camera->setBrightness(camera->getBrightness()-10);
54              break;
55            case Key::UP:
56              unsigned int max;
57              max = camera->getFeature(FEATURE_BRIGHTNESS)->getMaxValue(
```

```
58              camera->setBrightness(max);
59              break;
60            case Key::DOWN:   camera->setBrightness(0); break;
61            }
62          }
63          catch(Dyncon1394Exception& e) {
64            cout << e.what() << endl;
65          }
66
67          if (key.code()==Key::ESCAPE) break;
68        }
69        camera->getFastVideo();
70        camera->convertToRGB24();
71        console->load((void*)camera->getPictureBuffer()->getImage(),
72                      width,height,width*3,Format(24,0xFF,0xFF00,0xFF000
73                      Palette());
74        console->update();
75      }
76      camera->stopVideo();
77      delete control1394;
78      control1394 = NULL;
79    }
80    catch (Error &error) {
81      error.report();
82      camera->stopVideo();
83      delete control1394;
84    }
85    catch(Dyncon1394Exception& e) {
86      cout << e.what() << endl;
87      camera->stopVideo();
88      delete control1394;
89    }
90    return 0;
91  }
```

Appendix D

Dyncon1394 Reference

The Reference of Dyncon1394 is located in the doc directory of Dyncon1394.
See [DynAPI] for more information.

Bibliography

[Coriander] Coriander 1.0.0-pre3

[LibDC1394] LibDC1394 1.0.0

[LibRAW1394] LibRAW1394 0.9.0

[OpenPTC] OpenPTC

[ImageMagick] ImageMagick 6.0.6.2 C++-API

[Linux1394] www.linux1394.org

[sourceforge] www.sourceforge.org

[C++Prog] Biarne Stroustrup *Die C++ Programmiersprache*
Addison-Weslay Verlag 2000

[DynAPI] Dyncon1394 Reference

[IIDC] IIDC-Specification - 1394 Trade Association