Developing a Java API for digital video control using the Firewire SDK

Bing-Chang Lai, Phillip John McKerrow and Damon Woolley
School of Information Technology and Computer Science
University of Wollongong
bl12@uow.edu.au
phillip@uow.edu.au
dlw02@uow.edu.au

Abstract FireWire is a standard on the Macintosh platform, with most Digital Video cameras supporting the FireWire interface. We believe that it would be of great benefit to the Macintosh community to have a simple programming interface between the Macintosh platform and Digital Video cameras. This paper illustrates how to build a simple Java application that controls a Digital Video camera though the FireWire bus. This is the first step in providing a Java API for Digital Video camera interaction.

1 Introduction

FireWire is a high-speed bus (up to 400 megabits per second) designed to connect video devices and computers. This high speed, along with easy of use, hot-swapping and simplified cabling makes Firewire ideally suited for multimedia applications. Defined as IEEE Standard 1394-1995, Firewire has become a standard interface to Digital Video cameras. Firewire has been included with most Apple products for some time.

Digital Video (DV) is only one kind of data that can be sent across the FireWire bus. DV is a digital tape format for video cameras. Using compression similar to motion-JPEG, it stores video data at approximately a 5:1 rate. DV has a constant data rate of 36 megabits per second, making it easily transferred over FireWire. FireWire provides a mechanism that allows computers and DV devices to interact. On a DV camera, commands can be sent from the computer that allow for remote camera control. Video frames can also be delivered though the FireWire interface, allowing video from the camera to be sent to the computer screen, and video stored an a hard disk to be recorded to the camera.

We are building a Java Application Programming Interface (API) that will allow users to take control of a DV camera. This API will allow users to control a camera’s functionality, including retrieving frames from and recording frames to a camera, and also including the ability to retrieve time code information from the camera.

This paper describes a simple application that allows viewing and control of a DV camera connected to a Macintosh computer though FireWire. It is the first step towards building a fully functional API.

2 Software Architecture

The software architecture of this project consists of a C library that interfaces with the Firewire SDK. A Java wrapper class using the Java Native Interface (JNI) calls the C library. This allows applications written in Java to access DV controlling routines though the Java wrapper class. The application is built using Apple’s ProjectBuilder running

under Mac OSX. This application will only work using Mac OSX with a Firewire equipped Macintosh.

Control and Capture Application (Section 3)	Java
Java DV Controller (Section 4)	Java API
Java Wrapper Class (Section 5)	JNI
DV controller Library (Section 6)	C Library
DV Device control/ Isochronous data handler	IOFWCVComponents.kext
Firewire DV handler	IOFireWireDV.kext
Firewire SDK	IOFireWireFamily.kext
DV Camera	

Figure 1. Software Architecture

Figure 1 illustrates the layers of the system, from the highest level at the top, through to the lower level supplied by the FireWire SDK. The bottom row represents the physical layer, the Digital Video Camera. At the top, the Java Application makes an instance of the DV controller class. This class sets up an interface for sending commands to the camera. The controller class makes an instance of a controller in the JNI class. Calls to methods from the JNI class enact native C code, that is compiled as a shared library. This C library makes calls to a FireWire Digital Video Handler that is provided with the FireWire SDK 2.7 for Mac OSX.

3 Control and Capture Application

The Application developed enables users to remotely control a DV camera through an interface on the screen. The application also previews the video in a window on the screen. This application is written in Java, utilizing swing to provide the graphical user interface, and the Sequence Grabber in QuickTime for Java to provide the preview. The main Application makes an instance of the controller and the preview window as shown in the code below.

```
public class DVapp {
    public DVapp() {
        try {
            // make a controller
            DVController frame = new DVController();
            frame.initComponents();
            frame.setVisible(true);
            try {
                // make a preview window
                SGwindow preview = new SGwindow("Preview");
                preview.show();
                preview.toFront();
            } catch (Exception e) {
                e.printStackTrace();
                QTSession.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // Main entry point
    static public void main(String[] args) {
```

```
     new DVapp();
     }
}
```

The *DVController* class produces a window with all the buttons that allow remote controlling of a camera, as seen in Figure 2. The *SGWindow* class allows an on screen display of the camera's video.

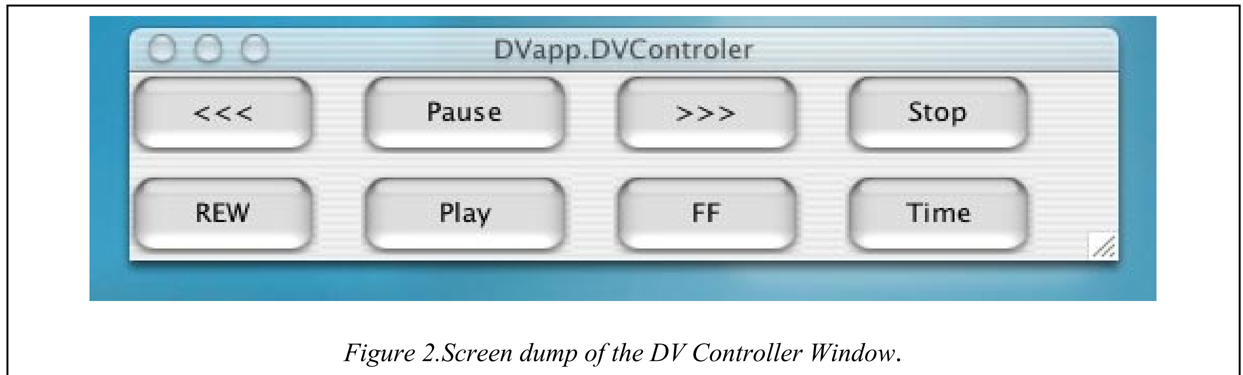

Figure 2.Screen dump of the DV Controller Window.

4 Java DV Controller

The *DVController* Class extends a *javax.swing.JFrame*. Members of this class are the *javax.swing.JButtons'* and the *JNIDV* interface class. When the *DVController* class is initialized the constructor makes an instance of the JNI native interface class *JNIDV*. The initialize functions from *JNIDV* are then called to allow control commands to be sent to the camera. The buttons in this window are members of the *DVController* class, they are initialized and an action listener is added to each button. When a button is clicked a *JNIDV* method is called. Below is the abridged code for the *DVController* class.

```
class DVController extends javax.swing.JFrame {
      javax.swing.JButton jButtonPlay = new javax.swing.JButton();
      JNIDV interface;
      public DVController() {
            interface = new JNIDV();
            interface.InitDV(); // call to native code
            interface.OpenDV();
            interface.OpenControlDV();
      }
      public void initComponents() throws Exception {
            jButtonPlay.setText("Play");
            jButtonPlay.setLocation(new java.awt.Point(110, 50));
            jButtonPlay.setVisible(true);
            jButtonPlay.setSize(new java.awt.Dimension(100, 40));
            setLocation(new java.awt.Point(5, 40));
            setTitle("DVapp.DVControler");
            getContentPane().setLayout(null);
            setSize(new java.awt.Dimension(470, 115));
            getContentPane().add(jButtonPlay);
            jButtonPlay.addActionListener
            (new java.awt.event.ActionListener() {
                  public void actionPerformed(java.awt.event.ActionEvent
e)
                  {
                        jButtonPlayActionPerformed(e);
```

```
            }
        });
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent
e) {
                thisWindowClosing(e);
            }
        });
    }
    void thisWindowClosing(java.awt.event.WindowEvent e) {
        interface.CloseControlDV(); // call to native code
        interface.CloseDV();
        setVisible(false);
        dispose();
        System.exit(0);
    }
    public void jButtonPlayActionPerformed(java.awt.event.ActionEvent
e){
        interface.controlPlayDV(); // call to native code
    }
}
```

The Preview window is instantiated from the main application. This Code is modified from the Sequence Grabber sample code on apple web site [1]. This Code is independent of the DV Control API that we are developing. A screen dump of both the DV controller and preview window can be seen in Figure 3.

5 Java Wrapper Class

As the application is written in java, and the Firewire API is a C library we have to develop an interface between them. Interfacing between the two languages is done using the Java Native Interface (JNI) to develop a wrapper class. This wrapper class allows a Java application to call functions from C code. Also it allows for passing objects between the two languages. The following code defines the Java side of the JNI interface.

```
public class JNIDV {
// The following Java methods call C functions
    public native void InitDV();
    public native void OpenDV();
    public native void OpenControlDV();
    public native void CloseControlDV();
    public native void CloseDV();
    public native void DVGetTime();
    public native void controlPlayDV();
// Load the JNIDV JNI library when this class is loaded.
    static {
        try {
            System.loadLibrary("JNIDV");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

When a program wants to use the DV control it must first make a call to *OpenDV()* to setup the device, then a call to *OpenControlDV()* to open the device for control transactions.

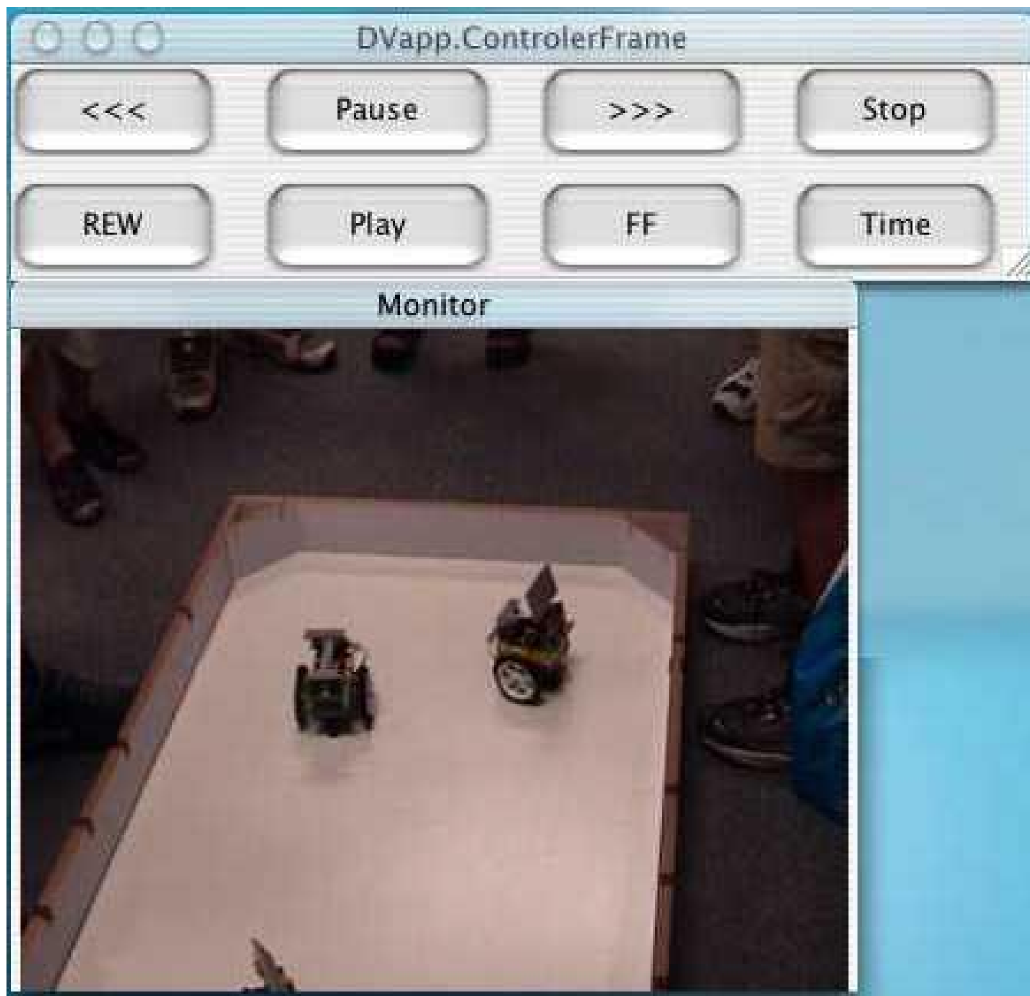

Figure 3.Screen dump of the controller and preview window.

The *InitDV()* prints a message to standard output indicating whether a camera device was found. Control functions such as *controlPlayDV(), controlStopDV(),* and *controlPauseDV()* can then be called to control the camera. Also, *DVGetTime()* can then be used to get the time code from the camera. The methods in this class call C code from *DV.c*, which interacts with the Firewire SDK.

6 C Library

The *DVComponentsGlue* framework is included with the FireWire SDK for MacOSX, the framework is for use with digital video devices and is documented in the SDK [2]. We wrote a C library to make the calls that interface with the *DVComponentsGlue* framework and the QuickTime framework. These functions calls are made by the C side using the JNI interface. The java wrapper class calls the public functions.

6.1 C Library Function Prototypes

Public Functions

```
JNIEXPORT void JNICALL Java_JNIDV_InitDV(JNIEnv * env, jobject obj);
JNIEXPORT void JNICALL Java_JNIDV_OpenDV(JNIEnv * env, jobject obj);
JNIEXPORT void JNICALL Java_JNIDV_OpenControlDV(JNIEnv * env, jobject
obj);
JNIEXPORT void JNICALL Java_JNIDV_CloseControlDV(JNIEnv * env, jobject
obj);
JNIEXPORT void JNICALL Java_JNIDV_CloseDV(JNIEnv * env, jobject obj);
```

```
JNIEXPORT void JNICALL Java_JNIDV_DVGetTime(JNIEnv * env, jobject obj);
JNIEXPORT void JNICALL Java_JNIDV_controlPlayDV(JNIEnv * env, jobject
obj);
JNIEXPORT void JNICALL Java_JNIDV_controlStopDV(JNIEnv * env, jobject
obj);
```

Private Functions

```
static void doControl(ComponentInstance theInst, QTAtomSpec
                              *currentIsochConfig, UInt8 op1, UInt8
op2);
static OSStatus notificationProc(IDHGenericEvent* event, void*
userData);
```

6.2 Description of Functions

InitDV:

This function finds the first component that is a camera using *FindNextComponent*. Then it prints the information about that component, which is obtained with a call to *GetComponentInfo*.

OpenDV:

This function first opens a *ComponentInstance* of the digital video camera device using *OpenDefaultComponent*. Calling *IDHGetDeviceList* and passing the *ComponentInstance* with a *QTAtomContainer* will return a list of devices. Making a call to *QTCountChildrenOfType* returns the number of DV devices connected to the computer. The code below illustrates these calls.

```
theInst = OpenDefaultComponent('ihlr', 'dv  ');
IDHGetDeviceList( theInst, &deviceList);
numberDVDevices = QTCountChildrenOfType
                 ( deviceList, kParentAtomIsContainer,
            kIDHDeviceAtomType);
```

The function then makes use of QuickTime calls to find the device and its configurations.

```
// get the atom to this device
deviceAtom = QTFindChildByIndex( deviceList, kParentAtomIsContainer,
                                     kIDHDeviceAtomType, i + 1, nil)
// find the isoch characteristics for this device
isochAtom = QTFindChildByIndex( deviceList, deviceAtom,
                                     kIDHIsochServiceAtomType, 1, nil)
// how many configs exist for this device
nConfigs = QTCountChildrenOfType( deviceList, isochAtom,
                                     kIDHIsochModeAtomType);

// get this configs atom
configAtom = QTFindChildByIndex( deviceList, isochAtom,
                                     kIDHIsochModeAtomType, j + 1, nil);


// find the media type atom
mediaAtom = QTFindChildByIndex( deviceList, configAtom,
                                            kIDHIsochMediaType, 1, nil);

QTLockContainer( deviceList);
```

```
// get the value of the mediaType atom
QTCopyAtomDataToPtr( deviceList, mediaAtom, true, sizeof( mediaType),
                                                      &mediaType, &size);

QTUnlockContainer( deviceList);

// is this config a video config?
if( mediaType == kIDHVideoMediaAtomType)   // found video device
{
      videoConfig.container = deviceList;// save this config
      videoConfig.atom = configAtom;
}
```

At the end of the *OpenDV* function, *IDHSetDeviceConfiguration* is called from the *DVComponentGlue* framework. *IDHSetDeviceConfiguration* sets a camera component instance to a specific configuration. This call is shown below.

```
IDHSetDeviceConfiguration( theInst, &videoConfig);
```

OpenControlDV:

The function *OpenControlDV* opens the device to enable it to handle action commands, such as play and stop. This function calls the *IDHOpenDevice* from the *DVComponentGlue* framework. *IDHOpenDevice* opens the currently configured camera. The currently configured camera is stored in the global *ComponentInstance* and passed to *IDHOpenDevice.*

```
JNIEXPORT void JNICALL Java_JNIDV_OpenControlDV(JNIEnv * env, jobject
obj) {
      OSErr err;

      err = IDHOpenDevice( theInst, kIDHOpenForReadTransactions);
      if( err != noErr)
            printf("error %d(0x%x)\n", err, err);
      printf("Opened device\n");
}
```

CloseControlDV:

CloseControlDV is called to close the device opened in the *OpenControlDV* call. It is similar to the *OpenControlDV* function. This function calls *IDCloseDevice* from the *DVComponentGlue* framework. *IDCloseDevice* takes an argument of the global *ComponentInstance* and closes the camera already opened.

```
JNIEXPORT void JNICALL Java_JNIDV_CloseControlDV
                                             (JNIEnv * env, jobject obj) {
      OSErr err;

      err = IDHCloseDevice( theInst);
      if( err != noErr)
            printf("error %d(0x%x)\n", err, err);

      printf("Closed device\n");
}
```

CloseDV:

This function makes a call to *CallComponentClose*. *CallComponentClose* takes the argument of a *ComponentInstance* and closes it.

```
JNIEXPORT void JNICALL Java_JNIDV_CloseDV(JNIEnv * env, jobject obj) {
    CallComponentClose(theInst, 0);
}
```

controlPlayDV:

There are a number of control functions, which can be called to send an action command to the camera. These functions call *doControl*, passing two *opcodes* to it. The *opcodes* passed by each function and a sample function call can be seen in the list below.

Action	**opcode1**	**opcode2**
Play	0xc3	0x75
Rewind	0xc4	0x65
Rewind Play	0xc3	0x4e
Fast Forward	0xc4	0x75
Fast Forward Play	0xc3	0x3e
Stop	0xc4	0x60
Pause	0xc3	0x7d
Next Frame	0xc3	0x30
Slow Forward	0xc3	0x35
Previous Frame	0xc3	0x40
Slow Rewind	0xc3	0x45

```
JNIEXPORT void JNICALL Java_JNIDV_ControlDV(JNIEnv * env, jobject obj) {
      doControlTest(theInst, &videoConfig, op1, op2);
}
```

doControl:

This function is passed a *ComponentInstance* of the DV camera and a *QTAtomSpec* reference to the video configuration. The other two arguments are the control commands. The first command specifies the group, e.g. 0xc3 is the play group, and 0xc4 is the wind group. Then next command specifies what kind of control it to be preformed from that group. The complete list of these parameters can be found from the document “AV/C Digital Interface Command Set” from the IEEE trade association web site [3].

The function *doControl* calls *IDHGetDeviceControl*, which returns an instance of a device control component, which was set by calling *IDHSetDeviceConfiguration* from the *OpenDV* call. The *IDHGetDeviceStatus* is called to get the device status from the specified configuration. Then the *DVCTransactionParams* are built up consisting of a command buffer pointer and a response buffer pointer. The function *DeviceControlDoAVCTransaction* is then called to send an AV/C command to the device and return a response from the device.

```
static void doControl(ComponentInstance theInst,
                   QTAtomSpec *currentIsochConfig, UInt8 op1, UInt8 op2)
{
    ComponentInstance          controlInst;
    ComponentResult            result;
    IDHDeviceStatus                  devStatus;
    DVCTransactionParams             pParams;
    char                             in[4], out[16];
    int                              i;
```

```
    IDHGetDeviceControl(theInst, &controlInst);
    IDHGetDeviceStatus(theInst, currentIsochConfig, &devStatus);

    // fill up the avc frame
    in[0]    = 0x00; //kAVCControlCommand;
    in[1]    = 0x20;
    in[2]    = op1;
    in[3]    = op2;

    // fill up the transaction parameter block
    pParams.commandBufferPtr = in;
    pParams.commandLength = sizeof(in);
    pParams.responseBufferPtr = out;
    pParams.responseBufferSize = sizeof(out);
    pParams.responseHandler = NULL;

    do {
        for(i=0; i<sizeof(out); i++)
            out[i] = 0;
        result = DeviceControlDoAVCTransaction( controlInst, &pParams);
        if(result == kIOReturnOffline) {
            printf("offline!!\n");
            sleep(1);
            continue;
        }
        if(result)
            break;
    } while(result != kIOReturnSuccess);
}
```

7 Conclusion

We hope this paper has helped make FireWire and Digital Video more accessible to the Macintosh community. Providing a simplified access to these technologies can have many applications such as in education, image processing, web-cams and the Internet, and monitoring and security. We plan to repackage this work as a "FireWire for Java" API to make it available to Java programmers.

References

[1]http://developer.apple.com/samplecode/Sample_Code/QuickTime/QuickTime_for_Java/SGCapture.htm

[2] The FireWire SDK can be obtained from: http://developer.apple.com/hardware/FireWire/Developer_Info.html#FireWireSDK The documentation is located in Examples/IsocTest/IsochComponentsRef.txt.

[3] http://www.1394ta.org/Technology/Specifications/specifications.htm