**User Manual**

***Compiling and Installation:***

Standard GStreamer and GTK+ packages were used for implementation. The application requires these particular plug-ins supported in your GStreamer environment:

*dshowvideosrc, dshowvideosink, jpegenc, jpegdec, ffenc\_mpeg4, ffdec\_mpeg4, avimux, avidemux, mulawenc, alawenc, vorbisenc, oggmux, directsoundsink, dshowaudiosrc, mulawdec, alawdec, vorbisdec*

***Audio:***

Recording

To record audio, first enter the desired bitrate into the input box on the right of the screen labeled Audio Bitrate. The default rate of 44100 will be set when you first launch the program. Next, click on the Record button in the Record Audio section. Go to the desired path in the prompt and enter the file name and extension you want. If you enter an extension that is not supported, you will get a get a warning. The audio types that are supported are raw (.raw), mulaw (.ulaw), alaw (.alaw), and vorbis (.ogg). Once entered, the program will begin to record from your default recording device. To stop recording, click the Stop button in the Audio Playback section to stop the recording and clean up the internal pipeline.

Playback

To playback an audio file, click the Record button in the Audio Playback section at the bottom of the screen. Go to the desired path in the prompt and enter the file name and extension you want. The audio types that are supported are raw (.raw), mulaw (.ulaw), alaw (.alaw), and vorbis (.ogg). If you enter an invalid file, you will receive a warning telling you that the pipeline failed to start. Once a valid file is entered, it will begin to play and you should hear it playing through your default playback device. When the file has ended or when you want to stop the playback, click the Stop button in the Audio Playback section to stop the audio and clean up the internal pipeline.

***Video:***

Recording

To record video from the camera, click on Record to the right of Camera Video label. A file dialog will appear for you to pick the destination of the recorded video (file extensions aren’t automatically generated). The settings such as resolution, frame rate and compression method can be input at the rightmost area of the GUI. There are no input checks, therefore width, height and rate must be numeric. The allowed methods for video compression are: MPEG4, JMPEG, or none. Changes in settings take effect on the next press of Record button.

Playback

To play a file, click on the Play button to the right of “Play Video File” label. A file dialog will appear to pick a file you want to play. Supported formats are .mp4 and .avi. The video will then run and at this time you may Rewind / Pause / Forward the video by pressing the corresponding button. You will then be allowed to pick a new video file when the current one ends by pressing Play again.

(In both features, you will be prompted of an error if invalid settings or files are being used.)

**Development Manual**

***Audio:***

The audio pipeline and all of its included elements are kept inside of a single data structure that is initialized in the main function and passed through the different functions that work with the audio pipeline.

Before creating the pipeline, the user inputs a file location and name along with a desired bitrate if they are recording. The filename is checked to see if the file’s extension is supported and returns the desired format if it is.

When creating a pipeline, it checks to see whether the source is a dshowaudiosrc element or if it is a file. If it is a dshowaudiosrc, it creates a recording pipeline using your default microphone. Otherwise, it creates a playback pipeline using filesrc with a location specified by the file name.

The recording pipeline creates a source element which receives the audio data from your microphone, a filter element that converts the audio data to the desired bitrate and number type (signed/unsigned integer or float), an encoder element that is selected based on the desired file type to compress the data, and, if you are saving the data as a vorbis file, a muxer element. Once it determines that all elements have been successfully created, it puts them all into the same bin, links them with the proper caps and sets the pipeline to playing.

The playback pipeline creates a source element filesrc with a location and file name supplied to it. It then creates a decoder element based on the file type if the data needs decoding, and a filter element to convert the decoded data to a numerical type the sink can understand. Finally it creates the sink directsoundsink which finds the default playback device and sends the audio to it. Once it determines that all elements have been successfully created, it puts them all into the same bin, links them with the proper caps and sets the pipeline to playing.

Stopping the pipelines uses the same function for both recording and playback. Inside the stop record/playback function, it sets the pipeline inside the audio data structure to READY, then sets it to NULL, and finally unreferences the pipeline used.

***Video:***

The video pipeline is separated into its own module where is gets the data it needs to know from the GUI module. Right after GUI and GStreamer initialization it is desirable for the camera video window to be overlayed on our own GUI, and so we have a callback that checks and uses the overlay capabilities of the initialized video sink if they exist.

The GUI has a state, where it starts from *Initial*, and transitions to *Camera* or *File* based on user interactions. These states help track which GStreamer elements are needed and which can be disposed of. When the application changes between states, a new pipeline is created for that particular state and its user-defined settings.

The pipeline for camera capture starts with the *dshowvideosink,* which provides us the video data from the camera, after which is converted to the right color space by an automatic converted, and then filtered based on the settings that users can manipulate (resolution, rate). Once the video data is in correct format we spawn two threads with one queueing data to the media player, which is directly displayed on the screen. The other thread queues the data to an encoder, based on the compression setting, which then passses it to the muxer for potential mixing with sound later on. After muxing, the data goes to a file sink component with location you specified when you pressed record, which will write the resulting file. If no compression is chosen, encoder becomes an Identity element which simply passes the data along the pipeline.

The pipeline for video playback starts with a file source which locates the file user speicifed. We then use a demuxer which for now only retrieves the video data, passing it on to converter to make sure the data is in the right colorspace, and finally the data reaches the video sink which get displayed in the GUI. The demuxer needs to be dynamically linked to the decoder, once its pads become available.

Video pipeline has its state PAUSE’d on Pause button press. Rewind and Forward commands generate seek events which manipulate the playback rate of the video accordingly. The camera pipeline becomes NULL once the recording has stopped, and transition between File and Camera player modes also cause the pipelines to become rebuilt.

No complex structures we necessary, audio and video components simply communicate with the GUI through structs. Different encoders can cause differences in the pipeline, therefore our code handles some of them separately.