

iPhone RTMP Library

Aftek Limited
50/24.Pralhad
Archad,Bhakti Marg,
Off Law Collage
Road,Erandwane,
Pune-411004
www.aftek.com

Confidential
V3

Disclaimer

The information presented in this document can only be used by prospect for the sole purpose of considering this proposal. It is deemed to be "commercial in confidence", and should therefore only be distributed to prospect employees or their authorized advisors. Material shall remain under the ownership of Aftek Limited and cannot be used for commercial benefit outside of this specific inquiry or project.

1. RTMP Library Functionality.

2. Installation Procedure.

3. Sample code.

3.1 Audio Playback.

3.2 Audio Publish.

3.3 Video Playback.

1.RTMP Library Functionality

Interface: RtmpLib

This interface could be used to perform different RTMP related operations such as playing the files.

rtmp_metadata getMetadata()

This function returns the Metadata object containing the meta-data information of the stream.

boolean isPaused()

This function returns true if the stream is paused or pause command is sent to the server.

boolean isConnected()

This function returns true if the stream is connected to the server.

int getRtmpStreamId()

This function returns the ID of the current stream that is returned by the server.

void closeRtmpStream()

This function should be used to close the RtmpStream created and release all the resources acquired by the stream. It will also be used to stop playing the file if the file is currently being played.

void connect(String url)

This function is same as above

void playRtmpStream()

This function can be used in two scenarios.

1. If user has specified the name of the stream/file on the server to play in the url, then user can use this function to start playing the file.

2. If the user has paused the stream, then it will resume the playback.

void pauseRtmpStream()

This function should be used to Pause the playing stream.

Interface: RtmpClient

Implement this interface to provide the client object to the stream created. The library calls back these methods on the provided object according to the server events. This is the communication point of library and application. The data coming from the server or the data to be published to the server is transferred using this client.

void onStatus(final String code)

This method will be called by the library when the server sends the status message.

void streamCreated()

This function will be called when the stream is connected successfully with the server. The connection with the server is two step process. Handshake of client and server.

If the handshake becomes successful, then the client requests the server to connect to the needed application. If the application is there, and client meets the needed criteria if any ,then server accepts the connection and responds with success.

When the server returns success, this streamCreated() function of the client object is called.

void onDataReceived(RTMPData data)[]

This function is called by the library to provide the actual data (Audio / Video / Metadata) of the stream/file on the server.

void onMetaDataRecieved(Map metadata)

This function will be called by the library upon receiving a meta data message from server.

Class: RtmpData

This class is a wrapper for actual RTMP data that is received from server. This class parses the incoming data to create a MetaData object for Audio and Video data.

byte[] getData()

This function returns the binary data corresponding to this Object.

MetaData getMetaData()

This function returns the MetaData object corresponding to this message.

Class: MetaData

This class is a meta data for an RtmpData object. Contains Audio/Video and time related meta data information.

boolean isMono()

Specifies if the audio data is Mono or not.

boolean isStereo()

Specifies if the audio data is Stereo or not.

int getSampleSizeInBits()

Returns the sample size of the audio data in bits.

float getSampleRate()

Returns the sample rate of the audio data.

AudioCodec getAudioCodec()

Returns the Audio Codec of the audio data as per FLV specs.

VideoCodec getVideoCodec()

Returns the Video Codec of video data as per FLV specs.

FrameType getFrameType()

Returns the Frame type of the video data as per FLV specs.

int getTime()

Returns the position in stream of the data.

Enum: RTMP_AUDIO_FORMAT

This is an enum of all the supported audio codecs. The various members of this enum are as follows.

FORMAT_PCM_PLATFORM_ENDIAN

FORMAT_ADPCM

FORMAT_MP3

FORMAT_PCM_LITTLE_ENDIAN

FORMAT_NELLYMOSER_16KHZ_MONO

FORMAT_NELLYMOSER_8KHZ_MONO

FORMAT_NELLYMOSER

FORMAT_G711_A_LAW_PCM

FORMAT_G711_MU_LAW_PCM

FORMAT_AAC

FORMAT_SPEEX

FORMAT_MP3_8KHZ

FORMAT_DEVICE_SPECIFIC

Enum: RTMP_VIDEO_FORMAT

This is an enum of all the supported video codecs. The various members of this enum are as follows.

JPEG

H263VIDEOPACKET

SCREENVIDEOPACKET

VP6FLVVIDEOPACKET

VP6FLVALPHAVIDEOPACKET

SCREENV2VIDEOPACKET

AVCVIDEOPACKET

Class: DownloadLimitExceeded(Exception)

This exception is thrown if the file or stream you are downloading extends the download limit of 2MB.

2.Installation Procedure.

1. ApplicationInterface.h and RtmpLib.h shall be included or referenced in the project.
2. click Projects \ Active Target Settings \ Build tab and set other linker flag to '-ObjC'.
3. RtmpLibSim.a shall be used to compile the application against simulator.
4. RtmpLibDevice.a shall be used to compile the application against device.
5. Open the source code file where RTMP library is to be used.
6. RTMP library shall be used with following steps,
7. Implement a class and implement RtmpLibDelegate protocol. Code for processing the received audio or video data shall be written.
8. Go to the location in code where RTMP data needs to be fetched (e.g. button click).
9. Create 'RTMPLib' object and call 'start' method with URL of RTMP server of that object.
10. These procedure is same for all the libraries(Audio/Video).

3. Sample code

3.1 Audio Playback.

Step 1 – Setting up connection with rtmp server...

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    rtmpController=[[RtmpLib alloc]init];
    rtmpController.delegate=self;
}

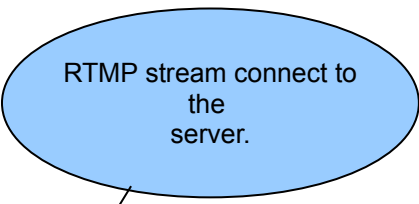
-(void)callRtmpMain
{
    NSAutoreleasePool *pool=[[NSAutoreleasePool alloc]init];
    rtmpStreamID=[rtmpController getRtmpStreamID];
    NSLog(@"RTMP StreamID :- %@",rtmpStreamID);
    NSString *rtmpUrl=[NSString stringWithFormat:@"%s@/%s",rtmpUrlTextField.text,rtmpFileNameTextField.text];
    char *charRtmpUrl=(char *)[rtmpUrl UTF8String];

    //[rtmpController start:"rtmp://172.16.3.133/test10/test_limit"
    streamID:rtmpStreamID];

    [rtmpController start:charRtmpUrl streamID:rtmpStreamID];

    [pool release];
}

```

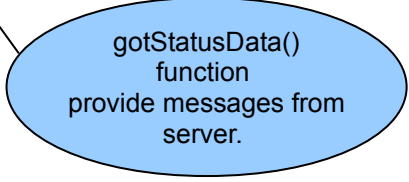


Step 2 – Handle response of the connection requests

```

-(void)gotStatusData:(NSString *)status_data
{
    NSLog(@"-----Status [%@]-----",status_data);
}

```



Step 3 – Handle metadata information of the stream. This is an optional step.

```

-(void)displayRtmpData
{
    rtmp_metadata data = [rtmpController getMetadata];
    NSLog(@"naudio_codec_id[%f] audio_data_rate[%f]
audio_sample_rate[%f]\n",data.audio_codec_id,data.audio_data_rate,data.audio_s
ample_rate);
    NSLog(@"naudio_sample_size[%f] duration[%f]
file_size[%f]",data.audio_sample_size,data.duration,data.file_size);
    NSLog(@"nframerate[%f] height[%f]
video_codec_id[%f]\n",data.framerate,data.height,data.video_codec_id);
    NSLog(@"nvideo_data_rate[%f] width[%f]
stereo[%f]\n",data.video_data_rate,data.width,data.stereo);
}

```

Step 4 – Handle received audio data received through stream. Typically in such cases a processing thread is started and is fed with audio data as it comes.

```

-(void)gotAudioData:(rtmp_audio *)audio_data dataSize:(long)dataSize
{
    aqc.sampleLen+=dataSize;
    memcpy(pcmdata+write_offset, audio_data->data, dataSize);
    if(!isFunctionCalled)
    {
        //playBuffer1(argStruct);
        [self performSelectorOnMainThread:@selector(callGotDataMethod)
withObject:nil waitUntilDone:NO];
        //self performSelector:@selector(callGotDataMethod) withObject:nil
afterDelay:2.0];
        //self performSelectorInBackground:@selector(callGotDataMethod)
withObject:nil];
        isFunctionCalled=YES;
    }
    write_offset+=dataSize;
}

-(void)callGotDataMethod{
    playBuffer1(argStruct);
}

```

Audio data provided by
server
Will play or save in file

```

int playBuffer1(myAQArgument argStruct)
{
    //static myAQStruct aqc;
    UInt32 err,bufferSize;
    int i;
    aqc.mDataFormat.mSampleRate = argStruct.sampleRate;
    aqc.mDataFormat.mFormatID = kAudioFormatLinearPCM;
    aqc.mDataFormat.mFormatFlags = kLinearPCMFormatFlagsSignedInteger|
kLinearPCMFormatFlagsPacked;
    aqc.mDataFormat.mBytesPerPacket = argStruct.bytesPerPaket;
    aqc.mDataFormat.mFramesPerPacket = argStruct.framesPerPacket;
    aqc.mDataFormat.mBytesPerFrame =argStruct.bytesPerFrame;
    aqc.mDataFormat.mChannelsPerFrame = argStruct.channelsPerFrame;
    aqc.mDataFormat.mBitsPerChannel = argStruct.bitsPerChannel;
    aqc.frameCount = argStruct.frameCount;
    //aqc.sampleLen = 0;
    aqc.playPtr = 0;

    err = AudioQueueNewOutput(&aqc.mDataFormat,AQBufferCallback,
    &aqc,NULL,kCFRunLoopCommonModes,
                                0,&aqc.queue);

    if(err)
        return err;
    NSLog(@"\n Frame Count is %d",aqc.frameCount);
    aqc.frameCount = FRAME_COUNT;
    NSLog(@"\n Frame Count is %d",aqc.frameCount);

    bufferSize = aqc.frameCount * aqc.mDataFormat.mBytesPerFrame;
    NSLog(@"\n Buffersize is %d\n",bufferSize);

    for(i=0;i<NUM_BUFFERS;i++)
    {
        err = AudioQueueAllocateBuffer(aqc.queue,bufferSize,&aqc.mBuffers[i]);
        if(err)
            return err;
        AQBufferCallback(&aqc,aqc.queue,aqc.mBuffers[i]);
    }
    err = AudioQueueStart(aqc.queue,NULL);
    if(err)
        return err;
    return 0;
}

```

Step 5 – Handle stop/pause action.

```
[rtmpController pauseRtmpStream];
```

3.2 Audio Publish.

Step 1 – Setting up connection with rtmp server...

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    selfObj=self;
    rPublish =[[RtmpPublish alloc]init];
}

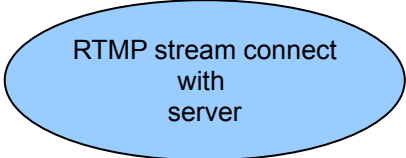
-(IBAction)startRecording
{
    statusLabel.text=[NSString stringWithString:@"Start Recording"];
    NSString *rtmpUrl=[NSString stringWithFormat:@"%s@/%s",rtmpUrlTextField.text,rtmpFileNameTextField.text];
    char *charRtmpUrl=(char *)[rtmpUrl UTF8String];

    [rPublish initRtmpPublishMain:charRtmpUrl];
    read_offset = 0;
    write_offset = 0;

    aqc.mDataFormat.mFormatID=kAudioFormatLinearPCM;
    aqc.mDataFormat.mSampleRate=22050.0;
    aqc.mDataFormat.mChannelsPerFrame = 1;
    aqc.mDataFormat.mBitsPerChannel = 16;
    aqc.mDataFormat.mBytesPerPacket=aqc.mDataFormat.mBytesPerFrame=aqc.mDataFormat.mChannelsPerFrame*sizeof(short int);
    NSLog(@"M Bytes per Frame %d",aqc.mDataFormat.mChannelsPerFrame*sizeof(short int));
    aqc.mDataFormat.mFramesPerPacket= 1;
    aqc.mDataFormat.mFormatFlags=kLinearPCMFormatFlagsBigEndian | kLinearPCMFormatFlagsSignedInteger | kLinearPCMFormatFlagsPacked;
    aqc.frameSize =2048;
    OSStatus status;

    status=AudioQueueNewInput(&aqc.mDataFormat,AQInputCallback,&aqc,NULL,kCFRunLoopCommonModes,0,&aqc.queue);
    if(status == 0)
    {
        NSLog(@"Audio queue created.....");
        char path[256];
        [self getFilename:path maxLength:size of path];
    }
}

```



```

        filename = CFURLCreateFromFileSystemRepresentation(NULL,
(UInt8*)path, strlen(path), false);
        status =
AudioFileCreateWithURL(filename,kAudioFileAIFFFType,&aqc.mDataFormat,
        kAudioFileFlags_EraseFile,&aqc.outputFile);
        for(i=0;i<AUDIO_BUFFERS;i++)
        {

AudioQueueAllocateBuffer(aqc.queue,aqc.frameSize,&aqc.mBuffer[i]);
        AudioQueueEnqueueBuffer(aqc.queue,aqc.mBuffer[i],0,NULL);

        }
        if(status == 0)
        {
                NSLog(@"MP3 file creates.....");
        }
        else
        {
                NSLog(@"Error in createing mp3 file.....");
        }
        }
        else
        {
                NSLog(@"Cannot Create Audio Queue %d",status);
        }

        NSLog(@"Start Recording.....");
        aqc.recPtr=0;
        aqc.run=1;
        AudioQueueStart(aqc.queue,NULL);

}

```

Step 2 – Handle publish ,audio data delivered through stream.

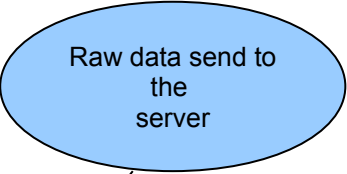
```
int encodeAudio(char * audio_data,int len)
{
    uint8_t outbuf[10000];
    int out_size;
    int outbuf_size = 10000;
    short * swap = audio_data;
    int i;

    for(i=0;i<len/2;i++)
        swap[i]=ntohs(swap[i]);
    out_size = avcodec_encode_audio(c, outbuf, outbuf_size, audio_data);

    [rPublish writeRtmpData:outbuf size:out_size rtmpPacketType:0x08];

    NSLog(@"****The Out Size is [%d]****",out_size);

    return 0;
}
```



Step 3 – Handle stop/pause action

```
-(IBAction)stopRecording
{
    statusLabel.text=[NSString stringWithString:@"Stop Recording"];
    NSLog(@"Stop Recording.....");
    AudioQueueStop(aqc.queue,true);
    for( i = 0; i < AUDIO_BUFFERS; i++)
    {
        AudioQueueFreeBuffer(aqc.queue,
                               aqc.mBuffer[i]);
    }

    AudioQueueDispose(aqc.queue, true);
    AudioFileClose(aqc.outputFile);
    [rPublish stopRtmpPublish];
}
```

3.3 Video Playback.

```
#import <AudioToolbox/AudioQueue.h>
#import <AudioToolbox/AudioFile.h>
#import <AudioToolbox/AudioServices.h>
#import "rtmpdump.h"
```

```
@interface PlayViewController : UIViewController<rtmpdumpDelegate> {
rtmpdump *rtmpController;
int rtmpid;
BOOL isFunctionCalled;
IBOutlet UIImageView *imageView;
float lastFrameTime;
int context_id;
NSTimer *videoTimer;
NSTimeInterval startTime;
UIImageView *displayImageView;
}
```

```
@implementation PlayViewController
@synthesize rtmpController;
```

Step 1 – Setting up connection with rtmp server...

```
-(void)viewDidLoad
{
    [super viewDidLoad];
    rtmpController=[[rtmpdump alloc]init];
    rtmpController.delegate=self;
}
```

RTMP stream to
connect
with the server

```
-(void)callRtmpMain{
    NSAutoreleasePool *pool=[[NSAutoreleasePool alloc]init];
    NSArray *paths =
    NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    NSUserDomainMask, YES);
    NSString* docDir = [paths objectAtIndex:0];
    NSString* resources = [docDir stringByAppendingString:@"./initFile.flv"];
    rtmpid=[rtmpController getRtmpStreamId];
    [rtmpController callRtmpStream:[NSString stringWithString:connectUrl]
    outputUrl:[NSString stringWithString:resources] rtmpId:rtmpid];
    [pool release];
}
```



```

-(IBAction)getStreamInfo{
    rtmpStreamInfo streamInfo;
    streamInfo=[rtmpController getRtmpStreamInfo:rtmpid];
}

```

Step 2 – Handle response of the connection requests

```

-(void)onStatusReceived:(NSString *)status
{
    NSLog(@"<-----Application Call Back--%@-->",status);
    [self performSelectorOnMainThread:@selector(displayImageViewMethod)
    withObject:nil waitUntilDone:NO];
}

```

Step 3 – Handle received audio data received through stream. Typically in such cases a processing thread is started and is fed with audio data as it comes.

```

-(void)gotData:(char *)rtmpData dataSize:(long)dataSize channel:(short)channel
sampleRate:(short)sample_rate sampleSize:(short)sample_size audioFormat:
(int)format;
{
    aqc.sampleLen+=dataSize;
    memcpy(pcmdata+write_offset, rtmpData, dataSize);
    if(!isFunctionCalled)
    {
        if(write_offset > 10560000)
        {
            isFunctionCalled=YES;
            playBuffer1(argStruct);
            [self performSelectorOnMainThread:@selector(callGotDataMethod)
            withObject:self waitUntilDone:NO],
        }
    }
    write_offset+=dataSize;
}

```

Raw data audio/video
provided by server will
play or save in file

Step 4 – Handle received video data received through stream.

```

-(void)callMethodInTread
{
    NSAutoreleasePool *pool=[[NSAutoreleasePool alloc]init];
    [NSTimer scheduledTimerWithTimeInterval:0.04 target:self
    selector:@selector(callRtmpGetVideoPacket) userInfo:nil repeats:YES];
    [pool release];
}

```

```

-(void)callRtmpGetVideoPacket
{
    NSAutoreleasePool *pool=[[NSAutoreleasePool alloc]init];
    [rtmpController setSpinFlag];
    [rtmpController getVideoDataFromQueue];
    [pool release];
}

-(void)gotVideoData:(char *)data dataSize:(long)dataSize
{
    //NSLog(@"\n-----Application Video Call Back----->\n");
    static int count =0;
    NSData *tempData=[NSData dataWithBytes:data length:320*480*3];
    [self performSelectorOnMainThread:@selector(displayVideo:)
    withObject:tempData waitUntilDone:NO];
    count++;
}

-(IBAction)displayVideo:(NSData *)tempData
{
    if(tempData!=nil)
    {
        UIImage *templImage=(UIImage *)[self
imageWithRawRGBADat1:tempData width:320          height:480];
        if (templImage !=nil)
        {
            imageView.image=templImage;
        }
    }
}

```