FlashyWrappers 2.55

User guide

1. Welcome to FlashyWrappers

FlashyWrappers allows you to capture videos of Flash / AIR apps locally and save them as files, to camera roll or send them somewhere over internet. You have full control over the video ByteArray and the video frames added. You can use FlashyWrappers for live stage recording as well as post-processing by adding video frames using BitmapData.

Supported platforms are desktop AIR (Mac / Windows), web (Flash Player) and mobile AIR (iOS / Android, which is still in alpha).

Supported video containers are MP4 (H.264 / AAC) for all platforms with the addition of OGV (Theora / Vorbis) solely for Flash as an alternative.

The interface is, in general, compatible accross platforms, though underneath different video encoding libraries are used, depending on the underlying OS.

2. Quickstart guide

1) Include the SWC or ANE for your platform into your project. **Flash Player (web) only:** copy FW_SWFBridge_ffmpeg.swf to your application root so it is together with your "main" swf.

2) Make sure to target **the latest** AIR SDK / **the latest** Flash Player runtime (19 at the time of writing this manual). Note: FW requires Flash 11.5 as absolute minimum, though some high level functions might not work 100% properly (but its workable).

3) Start coding!

Capturing **Flash / AIR stage** display object list.

1) ***Import FlashyWrappers:***

```
import com.rainbowcreatures.*;
import com.rainbowcreatures.swf.*;
import flash.events.StatusEvent;    // in case you're not importing this already
import flash.events.Event;          // in case you're not importing this already
```

2) ***Get FlashyWrappers class instance and setup event listener:***

```
var myEncoder:FWVideoEncoder = FWVideoEncoder.getInstance(this);
myEncoder.addEventListener(StatusEvent.STATUS, onStatus);
myEncoder.load(); // optional argument: path to FW_SWFBridge swf folder (Flash only)
```

3) Implement onStatus event handler & optionally start FlashyWrappers capture:

```
private function onStatus(e:StatusEvent):void {
        // FW is ready!
        if (e.code == "ready") {
                myEncoder.setFps(20);  // configure FW before calling start()
                myEncoder.start();
        }
        // encoder started, start capturing frames as soon as possible
        if (e.code == “started”) {
                addEventListener(Event.ENTER_FRAME, onFrame);
        }
        // video is ready!
        if (e.code == “encoded”) {
                var video:ByteArray = myEncoder.getVideo();
                // you can save to gallery on mobile
                if (myEncoder.platform == FWVideoEncoder.PLATFORM_IOS ||
                 myEncoder.platform == FWVideoEncoder.PLATFORM_ANDROID)
                myEncoder.saveToGallery(“video.mp4”, “My Album”);
        }
}
```

4) Capture frames on ENTER_FRAME event listener.

```
private function onFrame(e:Event):void {
        // capture the whole stage each frame
        if (phase == „record“) {
                myEncoder.capture();
        }
        // finished the recording, so call finish
        if (phase == „recording_finished“) {
                myEncoder.finish(); // now wait for 'encoded' status event
        }
}
```

That's it, you've captured your first video!

Low FPS & lags If your target platform is Flash Player, this is expected. In case you're trying to capture realtime videos you should use “realtime” mode (this is **on** by default). Check out the chapter called “Realtime mode” to understand why.

To optimize Flash Player recording speed, you need to encode at pretty low resolution in most cases due to Flash VM slowness. Reasonable is to encode at around **400x300 @20fps** on i5-i7 laptops to avoid raw frames accumulation in RAM (at higher resolutions the encoder would not keep up with the speed you're feeding frames). Use setDimensions(width, height) before calling start() to set the video resolution - try half the stage resolution.

OGV for Flash

At first, FlashyWrappers supported only OGV as video output format. FlashyWrappers was 100% FFmpeg based and all MP4 encoders for FW are problematic due to patents. Basically, all software encoders “attached” to your (commercial) software are subject to patents and possibly royalty payments.

That's why starting with 2.4, FlashyWrappers for AIR uses OS encoders (OS X, iOS, Windows, Android) instead of embedded FFmpeg encoders. This allows encoding MP4 videos without any legal issues as the encoders were already paid for by the OS manufacturers.

However Flash it not an operating system and doesn't expose its own MP4 encoders which could be used. So FW has to still “piggyback” FFmpeg encoders in the form of Flash SWF. You still have the option to use those MP4 encoders(OpenH264 and AAC to be precise), but dealing with the patents is your responsibility (or it should be the responsibility of the subject offering end user software).

If you wish to avoid this, OGV format offers a good compromise. No legal issues and similar performance to the MP4 format. You might transcode OGV to MP4 on your server after sending the video from Flash.

New MP4 ANE's notice

FW 2.4 introduces brand new OS X and Windows ANE's for MP4 encoding. Windows MediaFoundation and OS X AVFoundation are used and this might introduce new bugs, so please let us know if you encounter something strange!

3. Capturing Stage3D / DisplayObjects

If you pass no arguments to the *capture* method, it tries to capture the whole AIR / Flash stage. On desktop, this means capturing the complete display object list - naturally excluding Stage3D / StageVideo or similar layers.

On mobile, this means capturing *everything* including Stage3D / StageVideo or similar layers.

To capture Stage3D on desktop separately, you need to run “capture” with Context3D as argument:

```
// desktop
myEncoder.capture(myContext3D);
myContext3D.present();

// mobile
myContext3D.present();
myEncoder.capture();
```

On desktop, call this *before* myContext3D.present(). On mobile, call this *after* myContext3D.present().

On desktop, if you also want to capture the DisplayObject (MovieClips, Sprites...) layer on top (as you do on mobile by default), you need to call capture *twice*. You also need to init FlashyWrappers with *stage3DWithMC* set to *true*. This causes FlashyWrappers not to encode frame after you call *capture* the first time.

Two step Stage3D & MovieClip layer capture on desktop (Flash / AIR):

```
// setup FW for 2-stage capture
myEncoder.stage3DWithMC = true;
myEncoder.start();
...
// capture Stage3D
myEncoder.capture(myContext3D);
// capture Flash Stage (DisplayObjects)
myEncoder.capture();
myContext3D.present();
```

To capture individual DisplayObject, simply pass it as argument for capture:

```
myEncoder.capture(someDisplayObject);
```

This is supported both on desktop and mobile (not Android alpha) – but not recommended on mobile as its much slower than capturing fullscreen.

NOTE: FlashyWrappers wil automatically scale DisplayObjects to fit into the video dimensions.

4. Platform specific functionality

In general, FlashyWrappers is multiplatform, so you don't have to change your code. Given the differences between mobile and desktop platforms, 100% code compatibility is not always possible (such as method for saving to camera roll vs. saving file to disk).

FlashyWrappers is using a trivial method for detecting the platform its running at. The detected platform is available like this:

```
trace("The platform is: " + myEncoder.platform);
```

Possible string variables: ***IOS, ANDROID, WINDOWS, MAC, FLASH or equivalent FWVideoEncoder.PLATFORM_XXX constants.***

This way, you can have one code for all platforms with slight differences, for example you might want to encode at half resolution in Flash. If you use unsupported methods on wrong platforms FlashyWrappers will compile, but warns about unsupported methods on runtime in Flash debug console(or log), so it's always a good idea to launch in debug mode first and watch the trace log.

See the examples where platform dependant code is being used.

4.1. Mobile platforms: important difference

On iOS and Android alpha, when you use *capture()* method, the encoder uses OpenGL stage capture, which is much faster than if you captured individual DisplayObjects (this uses the "tranditional" capture method which converts DisplayObjects to BitmapData and sends those frames to the encoder as ByteArray).

When capturing in fullscreen, **do not** specify video resolution with *setDimensions* or in *start* and pass no arguments to capture():

```
myEncoder.start(20);
```

This tells the encoder to capture fullscreen at 20fps from OpenGL and set the resolution to match AIR stage. To capture fullscreen AIR frame from OpenGL:

```
myEncoder.capture();
```

This captures everything, including DisplayObjects, Stage3D, StageVideo on mobile.

4.2. iOS: Platform specific info

Minimum iOS version supported: **8.0**

Save video to gallery – call this method after getting 'encoded' status event. Calling this dispatches two possible status events: ***'gallery_saved'*** or ***'gallery_failed'***. The failed event usually indicates the user didn't allow the saving to gallery permissions for your app, so you might want to hint on that possibility in your response.

myEncoder.saveToGallery(filename, albumName);

Specifying filename causes the video to be ***copied*** inside the apps temporary storage folder. Be careful to not create too many videos with unique names to conserve the storage space. You can leave the filename string *empty(“”)* to let FW use the default video file directly.

albumName works for iOS8+ and creates the specified album if it doesn't exist yet & places your video inside. If you leave albumName empty it won't place the video into any album but still saves the video into the gallery.

Start audio mix – this works similar to *addSoundtrack* (mentioned in another chapter), but this method for adding soundtrack doesn't use AIR's “extract” Sound method - which might lag your app on iOS (if the soundtrack is long). Instead it uses the iOS mixer for mixing the mp3 natively in video post-processing phase. iOS_addAudioMix also remembers the time when you called it since the start of video encoding, so it adds itself at the time of calling it. So if you call this in the middle of recording, the soundtrack will start in the middle of the video (as expected).

Due an issue we've found, iOS incorrect reports mp3's length in some cases, so we added secondary “audio length” argument where you must pass the mp3 length to override the faulty length.

Don't forget to add the mp3 file as external file to your project(not just inside the project FLA) – in Flash CC you can include those files in Publish Settings. You can pass the “length” property divided by 1000(in case the MP3 is included in your FLA as well) or supply a known track length as second argument.1

*// **somefile.mp3** included as file, **somefileMP3** is the same track instantiated from a*
// Sound class

myEncoder.iOS_startAudioMix(“somefile.mp3”, somefileMP3.length / 1000);
myEncoder.iOS_startAudioMix(“anotherfile.mp3”, 23.52); // 23.52 seconds

Stop audio mix – you can stop the tracks added with iOS_addAudioMix at any time with this method. Again, call it whenever you need after starting to capture video and the audio will be stopped at the time of the call.

myEncoder.iOS_stopAudioMix(“somefile.mp3”);

Suspeding app on iOS

– FlashyWrappers on iOS automatically **cancels** the encoding if the app enters suspension, for example if home button is pressed, call is incoming etc. When this happens in the “recording” phase (when sending video frames to the encoder), you will be notified by '***stopped***' status event and you should properly reset your app's flow / UI to enable new capture session.

If this happens after “finish()”, which can currently take some time on iOS, you will be notified by '***encoding_cancel***' status event.

Distorted / damaged video

– when recording in non-realtime mode you might encounter distorted video when recording in resolutions the iOS encoder “doesn't like”. Its easy to test that effect 600x400 resolution – if you record in that resolution, the video will be distorted. If you record with 640x400 resolution, everything will be fine. We'll be investigating if this can be automatically corrected by FW, in the meantime try using “nice” resolutions (in general if you can divide the dimension by 16 it should be ok).

Photos permissions

– after calling myEncoder.load(), FW on iOS dispatches events which indicate whenever user has given your app Photos permissions. It also pops up a permissions dialog for the first time this is done. For more details & names of those events, check the API documentation.

4.3. Android: Platform specific info

Minimum Android version supported: **4.3**

FW 2.3 vs FW 2.4+ – there are very important differences between FW Android 2.3 (now deprecated) and FW Android 2.4. Here's a quick overview:

	FW 2.3	**FW 2.4+**
Based on framework	FFmpeg (Crossplatform)	MediaCodec (Android native)
API	2.3 (old)	current
addSoundtrack	Yes	Yes (since 2.45)
addAudioFrame	Yes	Yes (since 2.45)
Microphone recording	Yes	Yes
Android version supported	4.0+	4.3+
Current stability	High / device independent	Lower / device dependent
Speed (realtime possible)	Low(3-4fps) / No	High(30-60fps) / Yes
Video output format	OGV	MP4

FW 2.45 for Android is still very much in development. Android isn't great when it comes to device compatibility – what might work on one device might be broken at another device. Thats why we have implemented logging, including verbose logging mode. If FW doesn't work on your Android device, please send us two device logs – one using normal run and one using verbose logging mode. Note that FW 2.45 was rewritten considerably since FW 2.4 (mainly when it comes to code structure) so there might be additional bugs popping up between 2.4 and 2.45.

Audio & microphone recording Currently, there are still slight issues with audio on Android, which prevented us to use the “common” techniques working on all other platforms. However, we've implemented native microphone recording for FlashyWrappers. Which means, you can use the FWVideoEncoder.AUDIO_MICROPHONE mode to record video & microphone.

There are still some known issues even with this. For example, when verbose logging is turned out, audio frames might be lost due to the strain and as a result the video will be replayed faster than normal, with strange audio(thats because of the lost frames). Similar effect might be expected on lowend Android phones. In general, if you've got too fast video, its almost certainly because of audio issues.

To test that, try recording with AUDIO_OFF and see if your issue disappears.

With FWSoundMixer, there appears a problem to record microphone (using AIR's Microphone which normally works on other platforms). Other than that, you can use FWSoundMixer for Android to record everything as on iOS.

Save video to gallery – call this method after getting 'encoded' status event. Calling this dispatches two possible status events: ***'gallery_saved'*** or ***'gallery_failed'***. The failed event usually indicates the user didn't allow the saving to gallery permissions for your app, so you might want to hint on that possibility in your response.

myEncoder.saveToGallery(filename, albumName);

filename specifies the internal filename of the video(you might want to use unique filename or always different filename, depends if you want to fill the users device with multiple videos or just one / few being constantly overwritten). You can leave the filename string *empty(“”)* to let FW use the default video filename.

albumName creates the specified album if it doesn't exist yet & places your video inside. It is not recommended to leave this empty – FW forces Android to update its gallery to show your newly created album instantly. If no album is used it might be shown later, for example on the next device reboot.

Known issues

There are relatively many known issues for the new Android encoder. Some of them are connected to AIR, some are just symptoms of other errors / issues. We'll be trying to intelligently adress / avoid them in further releases(there was some progress in v2.45 towards that), but here are at least the most common issues you might face:

Recording not working at all

Apart from device quirks(currently MTK or Mediatek encoders seem to cause issues), this might happen if you start recording right after launching the AIR app. Ideally, put a timeout of 1-2 seconds and / or start to record after button click. The GLES context for AIR isn't immediatelly available and FW doesn't have reliable detection of its presence implemented yet. Basically, it seems AIR creates the context as soon as something is put on the stage (FW for Android tries to do it automatically at 1st frame, but we're not sure how reliable this is across devices).

Recording works, but freezes at the end

This is usually caused by some previous error which makes the encoder go into a bad state and culminates by the final freeze. Some of the most frequent issues we observed causing this:

– Starting to record right after adding Video with webcam to stage(might be true also for StageVideo or VideoTexture with webcam). Mixing up FW video encoder during webcam init seems to “damage” GLES state. Always wait a bit (timeout) after initializing webcam on Android before starting to record with FW. In device log you might see lots of these if you don't:

E/libEGL (16501): eglMakeCurrent:719 error 300d (EGL_BAD_SURFACE)

– Device too slow, audio recording is not keeping up - you will spot the following error in device log prior to the freeze:

W/AudioFlinger(172): RecordThread: buffer overflow

This causes the audio thread to not finish properly and the app to freeze. Try to record with AUDIO_OFF to see If your issue is that. **This error will likely happen also with verbose logging turned on, as the logging is slowing everything down considerably.** There can be number of other issues why the encoding didn't finish, and to know more we'll need your Android device logs.

4.4. Flash: Platform specific info

FW for Flash Player is a pretty special case on its own. In general, it is the slowest platform because it doesn't use any native code for video encoding (and we can forget about hardware acceleration as well). Instead, all the encoding is done in “AS3 on steroids”. You might have heard about “Alchemy”, “Alchemy2”, “FlasCC” or “Crossbridge”. Well, it is all essentially the same thing, compiled C/C++ code into AVM2 bytecode (which is what AS3 is compiled into as well). On steroids meaning that this code is better optimized and uses fast memory access (so called “domain memory”). FW for Flash is built on Crossbridge, the latest iteration of “Alchemy”.

However, video encoding on virtual machine is still pretty slow. Thats why Flash FW includes several techniques to work around the slowness.

Multithreading

This is automatically turned on in realtime mode – in non-realtime you can still use it, as explained in “Advanced” chapter. Simply put, all frames you are recording are sent to a background thread(Worker) where the encoder is processing those frames to not block your UI thread. There is a potential problem in case the encoder isn't keeping up (very possible if you're trying to record in something like 800x600 – not recommended!). The problem is: accumulating raw video frames in memory. The “raw” video frames are sent from Flash – but if the encoder isn't fast enough to eat them, they start accumulating in the RAM. This is of course a potential problem, and that is why FW for Flash employs the following technique...

RAM Guard

The FW RAM guard for Flash (introduced in 2.4) monitors the encoder performance in regards to the incoming raw frames frequency vs. encoder keeping up with it. If the encoder is not keeping up, it tries to lower the frequency of incoming frames by slashing down framerate – this is the simplest technique to make sure the encoder doesn't choke.

To prevent dropping frames, try to record in lower resolutions(max 400 x 300) or lower fps in general.

You can influence or even turn off the RAM Guard, but be careful. RAM Guard is not present on any other platform currently.

configureRAMGuard(RAMadjusterFPSDivider:Number = 2, RAMadjusterTolerance:Number = 1.5)

RAMadjusterFPSDivider *says what is the divider when RAM Guard needs to decrease fps. For example, if you are recording at 20 fps, but the encoder isn't keeping up, this fps is divided by 2, meaning the recording changes to 10 fps. If that doesn't help still, it changes to 5fps. And so on. All of this is happening in realtime. If you want the changes to be less severe, you would decrease the divider. If you want to turn off RAM Guard you'd set this divider to 1 (then the fps is never decreased as 20 / 1 = 20 all the time).*

RAMadjusterTolerance *says what RAM guard considers “slow encoder speed”. Basically, it watches how much the raw frames overflow the expected count. The default 1.5 means that if the buffer size is 150%+ of the expected buffer , thats a red flag. If you increase this number, the RAM Guard kicks in later.*

Flash plugin

Unfortunately, there have been issues with the Flash Plugin in the recent months which we couldn't influence. It is important to realize that because of the auto-update Flash plugin feature, a situation is possible where you test FlashyWrappers and everything is going fine, then you get back to it next month and it appears broken. We have learned 3 times in the past few months that this has been caused by Flash Plugin updates breaking Alchemy in various ways. *The last issue even crashed the Flash Player plugin entirely on regular basis. This has been resolved after intensive communication with Adobe.*

Luckily, AIR doesn't pose such issue because it is embedded with your application and can't auto-update. Therefore, its perfectly possible to ship your app with the latest stable AIR without being worried it would start crashing few months from now.

There might be other Flash issues not connected to FW, such as issues with hardware (various microphones, webcams etc.) - you should understand that we are not responsible for testing all possile hardware to be compatible with Flash. This is rather work for Adobe. We might forward any issues you are having with hardware to Adobe, but ultimately it is your responsibility to test everything with FW before running into deadlines etc.

4.5. Mac: Platform specific info

FW for OS X ANE has issues if you launch your app inside Adobe(and maybe other) IDE's. It should work when you publish a release build (.dmg installer). But inside the IDE, when launched, you will get an error stating that some ANE method couldn't be found(usally *fw_ffmpeg_create*). This is a bad way of AIR telling you that something is wrong with the native extension. A quick way to test whenever this is true is to publish to dmg. Keep in mind this can be also caused by other issues(such as not including the extension ID in the app .xml descriptor file – most IDE's do this for you). But on top of that, various IDE's such as Adobe Flash Pro / Flash Bilder *damage the ANE content* when it is unzipped into temp folder (they anihilate the symlinks and do other nasty things for some reason).

The workarounds usually involve one of those two(recommended is the second approach):

1) Unzip the ANE yourself into the IDE's ANE temp folder and make its folder read-only.

2) Unzip the ANE elsewhere - name the folder containing ANE files "*com.rainbowcreatures.FWEncoderANE.ane*". Then add **adl** launch argument inside your IDE (if possible) with **-extdir** "the folder containing *com.rainbowcreatures.FWEncoderANE.ane*".

As stated above, you really can just unzip the ANE's, they are just standard zip files internally.

5. Adding audio track

Adding a background audio track to your video is very simple on all platforms.

Right after getting 'started' event, you can mix a Flash Sound using addSoundtrack:

```
myEncoder.start(fps, FWVideoEncoder.AUDIO_STEREO, …);
myEncoder.addSoundtrack(new SoundFromLibrary());
```

Make sure to start encoder in AUDIO_STEREO mode.

6. Stopping and restarting capturing

In case you want to stop recording in the middle, whenever you want to finish properly or just cancel, you should always call “finish()” - this ensures cleaning the internals of FlashyWrappers.

Before recording again, call *myEncoder.start(...):*

```
function stop():void {
        // properly finish the current recording session
        myEncoder.finish();
}
```

7. Getting the encoding progress

You might need to display encoding progress, especially in multi-threaded mode when the encoder can works after the recording is over. Get progress by calling the following method each frame:

```
myEncoder.getEncodingProgress();
```

This returns (0 - 1), 1 meaning 100% done.

On iOS, you can keep calling this after finish(). iOS post-processes the video after finish() which takes a moment so FW starts to report new progress for this post-processing phase, until you get 'encoded' event.

8. Disposing the encoder

If you want to dispose the encoder and its C++ class instance, you don't need to do anything – the encoder and all buffers are initialized on “*start(...)*” and freed after calling “*finish()*”. That's why it is important to always call *finish()*, no matter if you plan on saving the video or not. When targeting AIR, you might want to call *myEncoder.dispose();* somewhere in the app's exit handler to dispose the context of the ANE.

9. FPS & audio syncing

In **realtime** mode, FlashyWrappers is trying to sync stage fps with video fps on all platforms starting with FW 2.4.

Simply put, if your app runs at 60fps and your video target is 30fps, FlashyWrappers automatically skips every second frame to keep up with the expected 30fps.

If your app runs too slowly (for example 15fps while the target video fps is 20fps), the video will simply be recorded “laggy” (ie at 15fps), even though technically its 20fps.

Note, In Flash(and possibly mobile), make sure you're not starting to record and asking microphone permissions at the same time. This will cause a delay in the audio.

10. Debugging, logging

To debug any issues with FlashyWrappers, try to turn on extended logging:

myEncoder.setLogging(FWVideoEncoder.LOGGING_VERBOSE);

Android

The log is present in general device log as well as in separate text file, usually in

/sdcard/Android/data/air.com.APPID/files/

iOS

The log is stored as part of your device log, and it is not present in any separate file

Windows

The log is in a separate text file in

C:\Documents and Settings\USER\Documents\FW_log.txt

OS X

The log is stored as part of your Mac log similar to iOS – you can access it by launching Console app from Applications / Utilities.

Flash Player

The log is stored in form of traces, so you will see it inside your IDE and / or in the Flash log file if enabled.

In case you encounter an issue, please try to look for a log without verbose mode first. Then switch verbose mode on, try to reproduce your issue and send that log as well. Sometimes, verbose logging can slow down the video encoding so much it might introduce additional issues, so its always good to have *both* standard and verbose logs.

11. Advanced

Realtime / non-realtime capture

FlashyWrappers allows two basic capturing modes: realtime and non-realtime.

As the name suggest, the default - **realtime** mode is useful when capturing live action (game, webcam recording and so on). This will make FlashyWrappers try to optimize its recording speed by employing more optimized fullscreen capture or multithreading if needed, or use other tricks such as frame skipping to make everything stay in realtime when playing the video back.

Realtime mode is switched on by default. You will need to turn it off in start method if you want to use **non-realtime**.

```
myEncoder.start(fps, audioMode, false);
```

If you want to implement some kind of post-processing, meaning you want to supply raw audio and video frames “offscreen”, non-realtime mode is for you. Non-realtime mode doesn't frame skip, it generally doesn't attemp to use fullscreen capture and also handles the pts in the video differently (pts = presentation timestamp, ie. when audio/video frame should be presented). In realtime mode the timestamp is calculated from the system clock to correspond to the live recording time, in non-realtime mode the pts is a hardcoded step, which doesn't depend on when the frame was actually encoded.

It is quite useful to know these differences when deciding which mode to use.

Capturing individual Display Objects on mobile

You can capture individual AIR display objects(including *Video*) on mobile similar to desktop:

```
myEncoder.capture(someDisplayObject);
```

This requires you to start the encoder in non-realtime mode(explained in chapter above) and also, you **must specify video resolution equal to the display object dimensions.**

If you attemp to capture displayObjects in realtime mode, FlashyWrappers will output an error, forcing you to capture in fullscreen, which is always the fastest possible mode.

You might want to override this behavior to capture individual Display Objects “in realtime”. FW is able to do that, though it won't be able to use the fullscreen OpenGL capture, which will **slow down** the capture. Keep that in mind and try to capture only smaller DisplayObjects on mobile. This approach works both for iOS and Android:

```
myEncoder.start(fps, audioMode, false, displayObjectWidth, displayObjectHeight);
myEncoder.forcePTSMode(FWVideoEncoder.PTS_REALTIME);
myEncoder.forceFramedropMode(FWVideoEncoder.FRAMEDROP_ON);

…

myEncoder.capture(displayObject);
```

Capturing rectangular area on mobile

As an alternative to the non-existing OpenGL target DisplayObject capture, you might use a feature introduced in FW 2.5 by using:

myEncoder.setCaptureRectangle(x, y, w, h, color, mode);

For details check the API documentation. This method only works on mobile, in realtime mode together with OpenGL capture. Additionally, in “calculate” mode this works only with stage scaling set to NO_SCALE and stage alignment TOP_LEFT. In “visual” mode it works with all scale and alignment modes but has its own drawbacks as it is trying to read uniquely colored pixels from FBO to measure the capture rectangle.

Capturing fullscreen(or using multithreading) for non-realtime

This is basically reverse to the technique above. Normally, capturing in non-realtime doesn't trigger the fast OpenGL fullscreen capture on mobile and doesn't enable multithreading in Flash Player.

However, what if you know you need to capture fullscreen in non-realtime mode? You could take advantage of the speedup techniques realtime mode uses while still generating “non-realtime” video with monotonic PTS & frame dropping off:

myEncoder.forcePTSMode(FWVideoEncoder.PTS_MONO);

myEncoder.forceFramedropMode(FWVideoEncoder.FRAMEDROP_OFF);

..

myEncoder.capture();

Modes for forcePTSMode: *PTS_AUTO, PTS_MONO, PTS_REALTIME*

Modes for forceFramedropMode: *FRAMEDROP_AUTO, FRAMEDROP_ON, FRAMEDROP_OFF*

Sending Bitmap frames to encoder (“offscreen capturing”)

In some cases, it might not be desirable for you to use *capture* method for sending video frames to encoder (especially when recording in non-realtime – you might want to encoder “offscreen”). Don't worry, FW allows you to send “raw” frames this way.

myEncoder.addVideoFrame(ARGByteArray);

This is possible on all platforms (including mobile). The ByteArray assumes ARGB(4) bytes for each pixel. If you don't know what it means, suffice to say that when you convert BitmapData to ByteArary using *getPixels(),* this is the format you'll get. Nothing prevents you from generating your frame pixels from scratch of course, filling the ByteArray and then sending that to FlashyWrappers. *With FW, you are not limited to “capturing” whats happening on the stage in relatime, you can make up video frames from scratch and send them whenever you need.*

Capturing microphone

Starting with FW2.4, there's a built-in microphone capture method. You must specify it as special audio mode in the start method:

```
myEncoder.start(fps, FWVideoEncoder.AUDIO_MICROPHONE);
```

Before calling start() and the built-in method will capture microphone together with the video. If you need to add soundtrack or other sounds with or without microphone, **do not** use this. Instead, use FWSoundMixer, which is part of FlashyWrappers SDK. FWSoundMixer allows microphone capturing as well as mixing sounds with microphone input from Flash, outputting stereo ByteArray.

Recording dynamic audio

In case you are trying to record sounds dynamically (for example, recording a game with soundtrack and sound effects), you might be wondering how to do that. Normally, you won't be able to capture any kind of raw sound data from Flash Sound mixer. Adobe doesn't provide any way known to us to do that in Flash Player.

To get around that, the only current option in Flash Player is using a custom sound mixer, for example our FWSoundMixer class. For AIR there are other (native) ways around this issue which we're working on.

In the meantime, FWSoundMixer allows you to mix multiple Sounds together in AIR just like in the Flash SoundMixer, by replacing your Flash “Sound” with “FWSound” class and also replacing your “SoundMixer” with “FWSoundMixer” class.

FWSoundMixer will play your sounds, but also provide a way to extract raw PCM audio from ByteArray and pass them to FlashyWrappers. FWSoundMixer is part of FlashyWrappers SDK, and you'll find it zipped inside the FlashyWrappers package.

If you want to provide raw audio data to FlashyWrappers, you can start in the following audio modes:

```
myEncoder.start(fps, FWVideoEncoder.AUDIO_MONO);
myEncoder.start(fps, FWVideoEncoder.AUDIO_STEREO);
```

Use AUDIO_STEREO if you're working with FWSoundMixer. The raw audio data are expected in floating PCM format(default for Flash / AIR and its audio methods), the ByteArray must be **little endian**. To send such audio to FlashyWrappers, you can call:

```
myEncoder.addAudioData(audioByteArray);
```

Typically, this is expected to be called as soon as you have audio data available – on most platforms the byte array can even contain a whole soundtrack(except Android right now).You can also send chunks of audio which you obtain from microphone for example & send them as you get them.

12. FAQ

Q: On Mac in Adobe CS / Flash Builder (or other IDE), an error says that 'fw_ffmpeg_create' from the native extension is undefined when launching my app from inside the IDE.

A: This is a pretty cryptic message, which might mean multiple things. On Mac the most likely cause is in the way the ANE was unzipped in temporary folder of your IDE. For reasons unknown to us, the IDE's don't unzip the ANE properly into its own temporary location(FB and CS6 like to do that both). We've worked in an internal fix, but it takes effect after second, third etc. build of your app possibly. You should try to find the temp folder, unzip the ANE yourself & make it read-only. Also try to publish .dmg installer to see if the app works at least like that or not. If yes, then the issue really is related to the IDE.

Q: ***I'm getting "Error #1065: Variable avm2.intrinsics.memory::casi32 is not defined." even when targeting FP 11.5+.***

A: You not only need to compile your project with the right Flash Player / AIR runtime version, but also launch it in the recent Flash Player / AIR version. Some IDE's (like Flash Player CS6) contain integrated Flash Player, which is unfortunately very outdated(usually version 11.2 or 11.4 after updates) and it cannot be updated.

You can update the "debugger" player, which can be launched by using CTRL+Shift+ENTER.

To do that, go to Adobe's website and download the latest "debugger" player (presently called "Projector content debugger", EXE) https://www.adobe.com/support/flashplayer/downloads.html.

Then overwrite Flash Debug players in your Flash CS installation, in the locations described here for example: http://stackoverflow.com/questions/15089375/location-of-flash-player-exe-used-by-flash-professional-cs5-cs6-ide-to-test-mov.

If you launch FlashyWrappers project in your Flash CS IDE, by using Ctrl + Shift + ENTER, the error should disappear. If you launch using Ctrl + ENTER, the error will still appear, because you're launching the integrated 11.2 / 11.4 Player.

Q: ***I'm getting PackageVerificationFailed or similar errors when trying to publish the examples to iOS / Android / Windows (likely in Flash CS6).***

A: Quit your IDE. Go to the "bin" folder of the example and erase the "...-app.xml" file. Restart your IDE and try to publish again. When publishing to multiple AIR platforms, Flash seems to get confused and is trying to use XML file from different platforms.

Also, make sure the iOS "App ID" is in accordance to what you've set on Apple's provisioning portal. You might have used your domain ID, such as "com.yourcompany.*", but after erasing the XML file this might have disappeared. In general, Adobe IDE's can get crazy with using cache. Always try to clear all temporary files, including the ones in Documents (contains unzipped extensions etc.).

Q: ***Can I play the OGV video inside Flash Player for preview?***

A: Not presently – Flash / AIR doesn't support Vorbis or Theora codecs. We will create a custom OGV player for this probably. Currently we've phased out OGV and its staying only in **FW for Flash Player**. All other platforms support mp4, which can be replayed natively in Flash / AIR. Integrated replay will come in the future.

13. Legal

FlashyWrappers 2.4+ for Flash currently use FFmpeg. To comply with FFmpeg's LGPL 2.1, users of your app basically need to be able to replace FFmpeg with new versions of the library if needed. This can get annoying for you if the libraries you're using are not linked dynamically.

Flash Player (web): we'll provide object files upon request which you can link / distribute to your customers in case they need to relink FW_SWFBridge_ffmpeg.swf.

All other platforms have native video encoders in use(not FFmpeg), so no action on your part is needed.

IMPORTANT NOTE: FlashyWrappers for Flash Player consists of OpenH264 / AAC codecs and MP4 muxer. The implementations, although they are not GPL (meaning you don't need to provide source code of your app as you're not bound by GPL) are still based on patented technologies. MAKE SURE YOU UNDERSTAND THE LICENSING TERMS OF MP4, OPENH264 AND AAC BEFORE USING THEM IN YOUR END PRODUCT. BY INCLUDING FLASHYWRAPPERS SDK INTO YOUR FLASH PROJECT, YOU AGREE ANY ROYALTY PAYMENTS FOR H.264 / AAC OR MP4 ENCODERS USAGE IN YOUR END PRODUCT ARE YOUR RESPONSIBILITY AND NOT THE RESPONSIBILITY OF THE AUTHORS OF FLASHYWRAPPERS.

Note: The above doesn't naturally apply to FlashyWrappers for Flash using OGV(Theora / Vorbis) encoders. Those codecs are completely free. IF you want to be “safe” when using FlashyWrappers for Flash Player, always used OGV encoders.