Design of Video Sync

on direct files from disk and/or web servers

This document describes the essentials of the basic code of Video Sync implementation in javascript with examples. It explains the basic concepts on how it works, some of the terminology we are using and why. We will discuss in short the following items goals, test video files, speed control, masterclock decoupling, metadata storage and transport latency compensation. At the end we will also tell you to find the demo, download, how to directly use them and the demo-video we will also share.

Goals

The main goal is to be able to sync any basic html video player with a master clock with a max of 100ms during playout and to be frame precise when paused. We want a system that can be extended easily so multiple clocks need to be possible and each of these videos should be able to sync to any clock we assign to it too. As a test we showcase an example with 4 videos (quad view) that are synced in this matter. It needs to be easy to merge this example into any framework so a good way to decouple the clocks from the video elements is needed. It is also important that syncing is done in the background meaning the viewer should not really know its going on syncing on startup/pause, seeking and drift during longer playout should be compensated with as little effect on the playout perception as possible.

Example Clocks

In the example we supply 4 specially created video files (different colors, stream names) to be able to show the process of syncing the analog clocks work as a stopwatch showing different time frames.

A analog minute dial and a 30min view this is much easier on the human eye to follow, The digital part is more interesting since it consists of both rendered time and timecode. The white time is the time we rendered into the video and will be in sync with the analog clocks. The colored (in this case blue) numbers are added by a NLE (in this case final cut pro) that was used to edit the videos and we added 2 numbers: the frame number of the video and the timecode of the video. The timecode and digital (white) video timer should be very close but not always the same since when making any video of a clock it's nearly impossible to render something frame precise but we should always see they are within the same video frame (in this case 1/60). This slight difference will only be seen in a video with a clock inside, not on other content and for the syncing we will look at the colored values.

The goal is to get all videos on the same ‘frame’ (and so same colored timecode). But keep in mind most and also the one in this example timelines will use time (in milliseconds) and not frame numbers as its units so we can and will see the same very small differences between the timeline and frame/timecode numbers. A frame based timeline would be possible but that is not normal given that other signals / data during the capture (like eye tracking for example) will always be time

based and not video frame based. The videos are large (about 1 Gig per video for 30min but please don't reencode them for lower quality since we can't make any claims anymore on the framenumbers after the reencode).

Speedup / Slowdown of video

The core trick we are using in this video syncing is keeping track of the difference between the master-clock and the video time and if the difference between them gets too big to speedup or slow down the video so the gap in time becomes smaller. We vary the speedup and slowdown depending on the gap size. To be able to see this working we programmed a range from 2x to ½ speed so it's easy to see the corrections in action but combined with a good 'seek' setting (discussed next). The obvious question is "*do people notice the speedup/slowdown of the video?*" The answer is when done right : *no* and it's also what happens in real life video playout (and clocks in general) always 'drift' in that over time their speed is too high or too low. Also by design to solve some technical issues (like PAL vs NTSC) it was very normal that whole movies would be played 4% faster in europe (PAL) vs the united states (NTSC) same with many radio stations where it's not uncommon to speedup playout (old pitch) by 2%-3%.

The way we use speed control is different. We are only doing a speed up or slow down over a small period of time and basically to fix the slow or fast playing of a video in the first place (drift from the master clock). The only other method would be to seek (jump) to the correct time until we get the video 100% correct and since a seek also has a slight over or undershoot (for example based on keyframes) that would also add some imprecision. So in the end we will control the gap between the assigned masterclock and the video time with both seek (jumps) and speed control. The seeks are more visible when the gap was big but the speed control is hard to notice when we use real values instead of test values (to showcase the effects).

Seeking on startup, scrubbing and gap correction

Seeking in video is the technical term for jumping in a video by a video player. This sounds like a good trick to get videos in sync if a video is not at the same time/frame, let't then just jump to the correct time. But that won't fully work and will be seen by the viewer much more also a seek (jump) is much heavier for a video player to perform and as a result will take time how long it takes is not stable so will introduce a gap by itself. So if you just seek to a new point and if any difference between the video player and the masterclock is still available seek (jump) again will often put you into an infinite loop of jumping around or/and be very visible to the viewer. As a result we do use seek but only when the gap is large enough to get bigger gaps down.

So the idea is that the gap is monitored and when it's bigger than say 2 seconds a seek (jump) is performed then we will see how small the gap is and the speedup/slowdown method will take over to bring it into sync again. This means that the end user will hardly notice. Do keep in mind in this demo/video presentation we kept the gap for jump very high (5 seconds) to show how it works this should be much lower on production code. The argument on that a seek taking time is very clear if you say have 8 videos on one clock where a seek (jump) needs to be performed on all 8 at the same time. Depending on speed of the computer you will easy see the seeks taking time and that after the seek we are in the ballpark of the time we need but small corrections using the speedup/slowdown method is still needed. Scrubbing in video on the timeline is basically a seek every time interval but once it starts up the videos (depending on if it was playing or by unpausing) you will also see small corrections after the last seek. This small difference can have several reasons for example the keyframing of the input videos that in this case are 50p and seeks mostly happen to keyframes.

Masterclock decoupling

The speed control and seek combined are the core of the concept but we also decided to decouple the defined clocks (timelines) from the videos. Normally you could directly link (with something like a subscribe) a video to the clock directly

but in practice that creates some issues and more importantly limits how easy this can be integrated into multiple frameworks or even server based clocks. Within many frameworks even if all the visuals are shown in one page to the user it's likely that within the framework things like timelines and video panels are on different parts/files/classes/objects. Even if we would directly link timeline to videos it would give small issues like when you scrub you will get too many events and need to debounce (delay) most requests. For this reason in this example we 100% decoupled the clocks from the rest of the system.

We write the clock state to some memory on regular intervals and this memory is then followed by the other parts of the software that are interested in this clock. All frameworks have ways to store the state and in this example we have used a browser storage but it will be easy to store it somewhere else once the framework is clear. The result is a highly portable way to port it to other javascript based frameworks and allows any number of clocks to be defined and any number of other things to follow such as videos but also other parts that are interested in this clock. They can be defined by name and any part of the software can access the clock state by that name and use it for what is needed. In this example it's used in the video pages and debug pages. In Fig.1 you can clearly see the decoupling where the timelines create/maintain the clocks (green), save their state into the storage (blue) and subscribers (red video, purple debug).

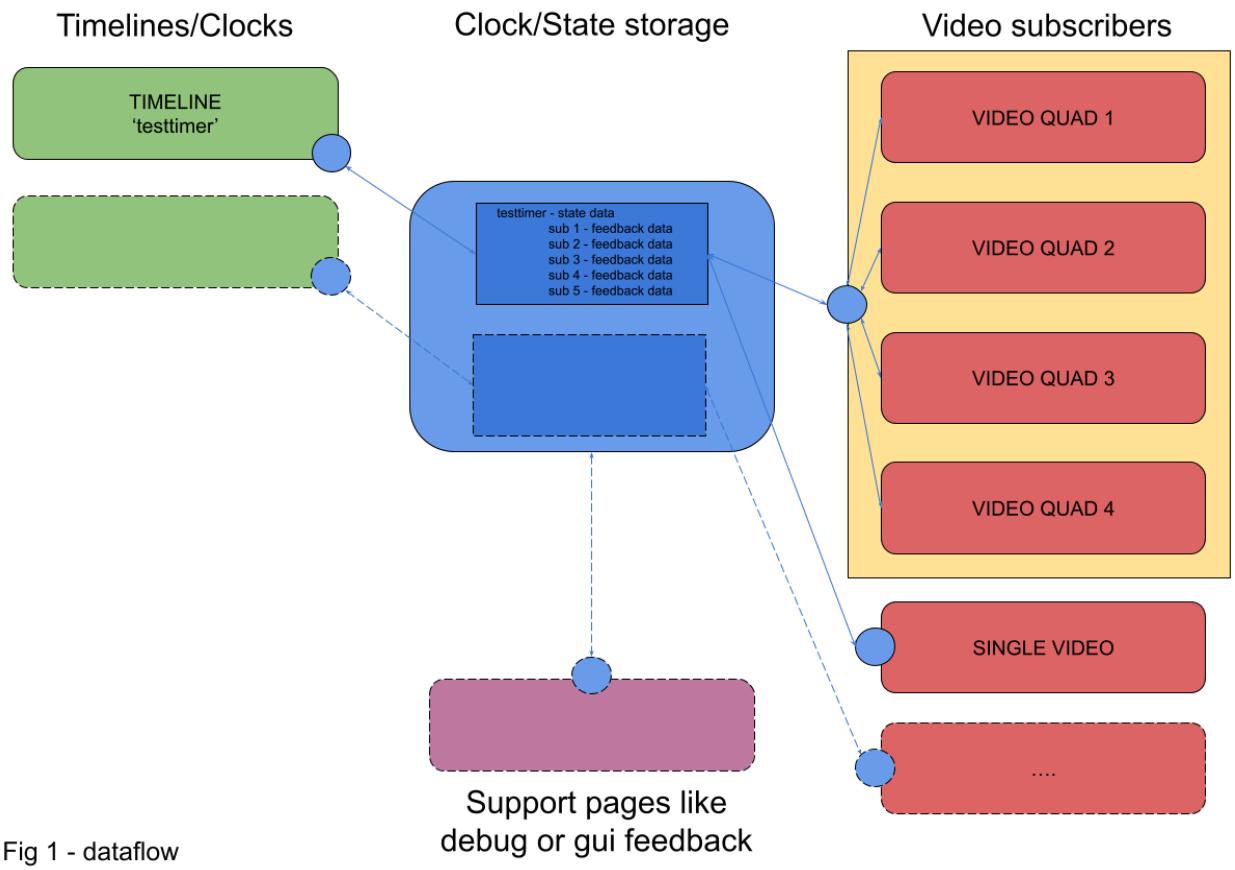

Fig 1 - dataflow

Metadata storage

If we look at Fig.1 you see we are saving more metadata into each clock storage (based on its name). This is one of the great side effects of the decoupling of clocks from video's / other users of that clock. Since we already defined this storage (we now do it as a json) it's very easy to add more information about the clock for anyone who is using this clock. In this example we added a few fields to both the clocks and the subscribers but this can easily be extended. For the clock we can for example store the name, timestamp, seek mode, running state and more. For the videos subscribing I put as an example current time the video is at, last time it synced, current (correction) speed, media src, duration, width, height and a few more but again it's easy to add other values that can then be picked up

by other parts if needed. See the source comments for all the fields we added and why.

Transport latency compensation

Next to the time gaps created by seeks and drift we also have the problem of delays created by transport times in the system. And it is good to notice that we make this worse by decoupling the clocks from the videos. We do this because we have no idea how quick the storage will be and how heavy the load will be if we sync more or less often. Currently we are running the clocks on and videos on one system but you can see how if we would run the clocks and videos on different machines servers will be involved and transport delays would be even more of an issue.

We also want to make sure that if messages from a clock get delayed or go missing the videos will keep going as best as they can. The idea is that messages from the masterclock are only updates on where the videos need to be but if some instructions are missed the worst thing that can happen is that the video drifts more than when we do get these updates. So the way this is compensated for is that each store moment of the clocks also saves a timestamp when it was done and once a subscriber used it (say a video) it also keeps a timestamp on when it was received so we know how old the message is. This is then used to correct the time in the incoming message so we know where the clock is at the moment of receiving as best as we can (even if the message is 2 minutes old, it should still be running at that time + 2min). This doesn't mean that we don't want to update the clock memory often to keep correcting the videos as best as we can but if some random latency or even fixed latency will not affect the end result much.

Online demo, download and video-presentation

Video-presentation : Its best to watch us showing how to use the online demo or download before using them :
http://test.noterik.com/noldus/video_sync/presentation.mp4

Download : We made a demo that can be downloaded and used on your own computer it can be downloaded at test.noterik.com/noldus/video_sync/video_sync_15Jul.zip You can download it and open the index.html file directly on your computer.

Online demo : the online demo can be used at :
http://test.noterik.com/noldus/video_sync/index.html you will always find the last update here but do watch the video presentation on how to use it.

Questions and feedback

Daniel Ockeloen - daniel@noterik.nl
Pieter van Leeuwen - pieter@noterik.nl