

Currently GStreamer will try to create several duplications of CODEC instances even if the types are all the same. For some applications such as an embedded hardware CODEC which could be accessed only by one decoding thread, it will be an issue.

It seems that decode bin version 3 has tried to resolve this problem. But for now, I still struggled in this issue. In order to support adaptive streaming (DASH) which has multiple adaption sets (ex, native YouTube has dual video tracks as well as dual audio tracks although the types are not same), I need a way to resolve this problem.

Here I try to insert input selector between demuxer & codec (in fact, between the multiqueue right after demuxer & codec) to resolve this problem. It seems to successfully achieve the requirement of a unique codec. I still work on the task to see if do switch between tracks is O.K but would like to share my current implementation here. I will be very appreciated if anyone has better idea and could give me a feedback or suggestion.

A given testing DASH URL below contains dual audio tracks with the same type (audio/mpeg, mpeg version = 4). It is a good point to start the experiment so I worked on it.

<http://marela.tilab.com:8085/content/dash/clear/multiaudio/2audio/c1/all.mpd>

command =

`gst-launch-1.0 -v playbin`

`uri=http://marela.tilab.com:8085/content/dash/clear/multiaudio/2audio/c1/all.mpd`

`flags=0x56`

Firstly the comparison is given here. You could see the new trial contains only a unique instance of audio CODEC but the older has duplications.

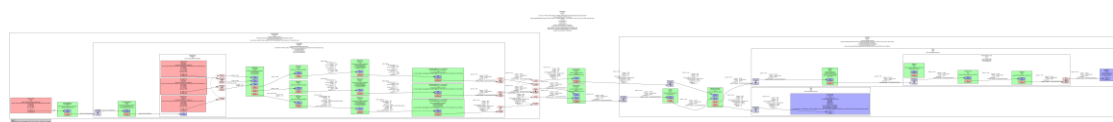


Fig 1 Native flow

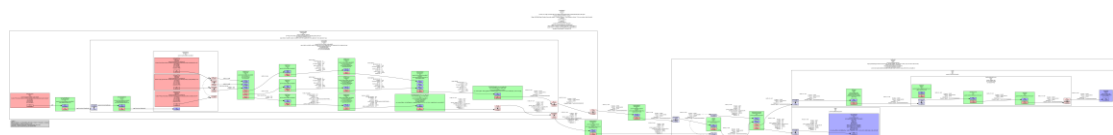


Fig 2 with input selector

One could see the difference of framework by the comparison. With native flow I could not do playback by the limitation of multi-thread. After connecting input selector, it is O.K (do switch between tracks is still in progress).

Please refer to the patch based on 1.6.1.

We list the overall flow chart here in advance.

The implementation locates mainly within connect\_pad().

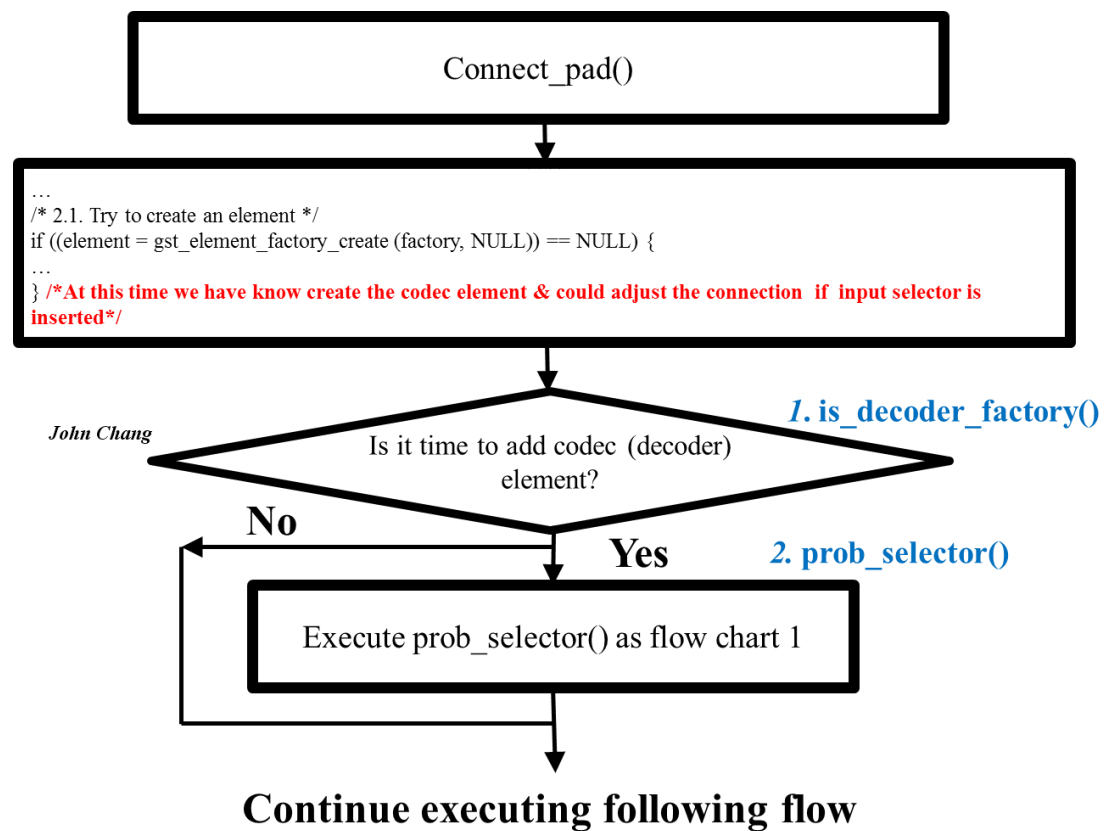


Fig 3 the Overall design

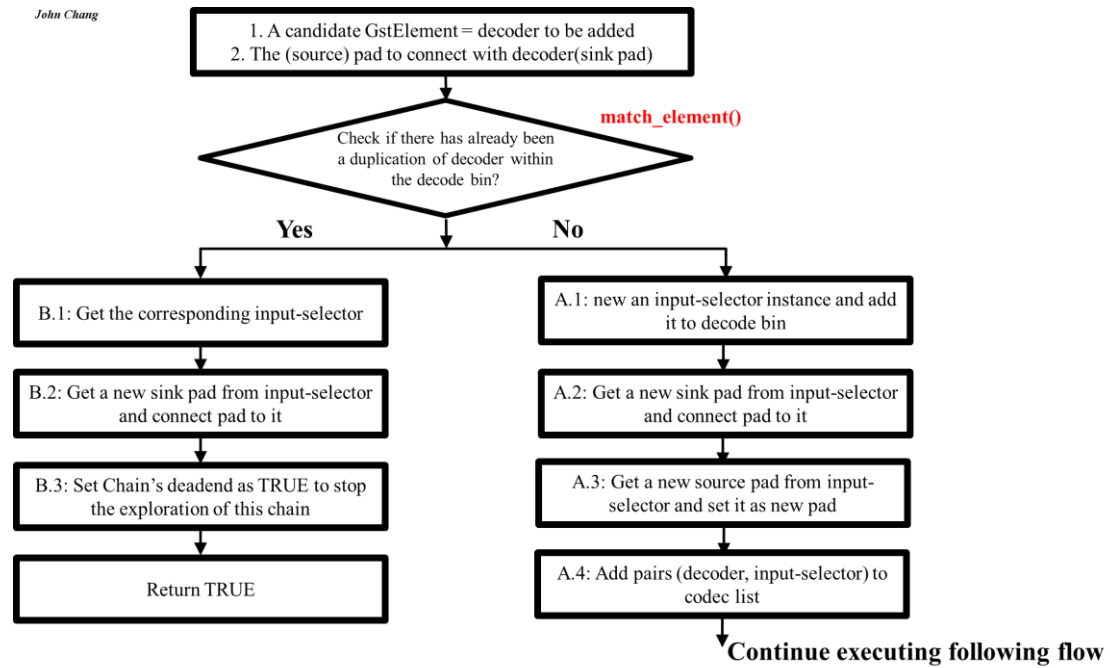


Fig 4 flow chart 1 – prob\_selector()

The figure below shows the snapshot when decode bin is about to add the 1st audio track's decoder.

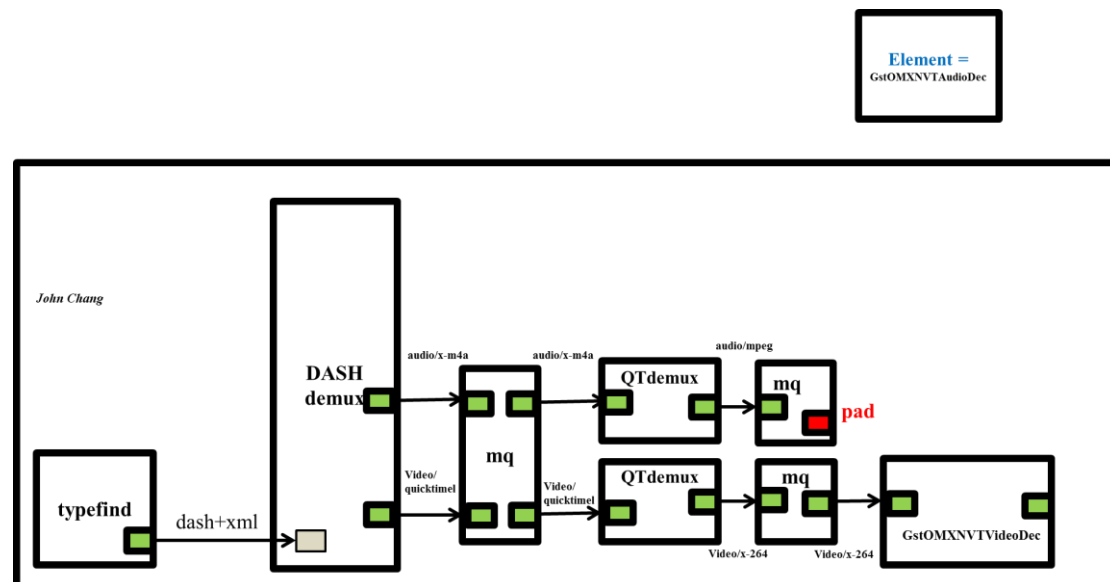


Fig 4 when decode bin is about to add the 1st audio track's decoder

First we analyze QT demux's source pad when the pad\_added\_cb() is called. Since it is a demuxer, we need to append an additional multiqueue. The QT demux will identify the output caps as "audio/mpeg" & mpegversion = 4. Through searching the

corresponding factories by caps, we find the target = OMX audio decoder. As the result, when we are in connect\_pad() & about to do connect, the source pad of multiqueue (in red) will be connected to the sink pad of OMX decoder in native flow.

Once the unique codec setting has been turned on, we need to insert input selector between multiqueue & OMX audio decoder. It is because OMX audio decoder is the first element we insert with such type (Gtype).

Here the requirement arises as we need a way to know if it is the first instance of a kind of codec element. It is achieved by function = **match\_element()**.

**match\_element()** complete the task by checking the element type of GstElementFactory since different factory has different element type and makes different type of element.

Also, we need to know if it is the time to insert codec element; it is achieved by function **is\_decoder\_factory()**. Function **is\_decoder\_factory()** does judgment by classification.

At this stage match\_element() return NULL so an input selector will be inserted, following the steps below:

A.1: New an input-selector instance and add it to decode bin.

A.2: Get a new sink pad from input-selector and connect pad to it

A.3: Get a new source pad from input-selector and set it as new pad

A.4: Add pairs (decoder, input-selector) to codec list

Please check the figure 4's path with prefix = 'A'.

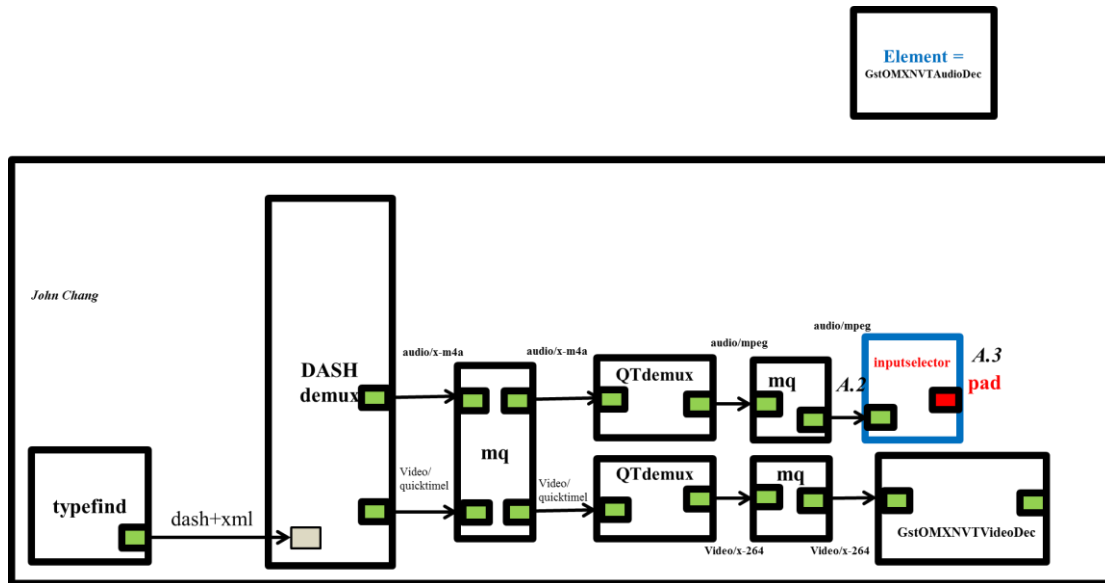


Fig 5 after inserting input selector

After completing connection, we replace pad = multiqueue's source pad by input selector's source pad. It makes no difference from CODEC's point of view since it's sink pad still will get connection to a source pad no matter the source pad is from mq or input selector.

Finally, we append the pair (type, input-selector) to a list (named codec list of dbin). It helps us to check if the same type CODEC has already installed & connect to corresponding input selector if it has so.

Now turn to the case when the second audio track is about to install its CODEC. The figure below shows the snapshot when.

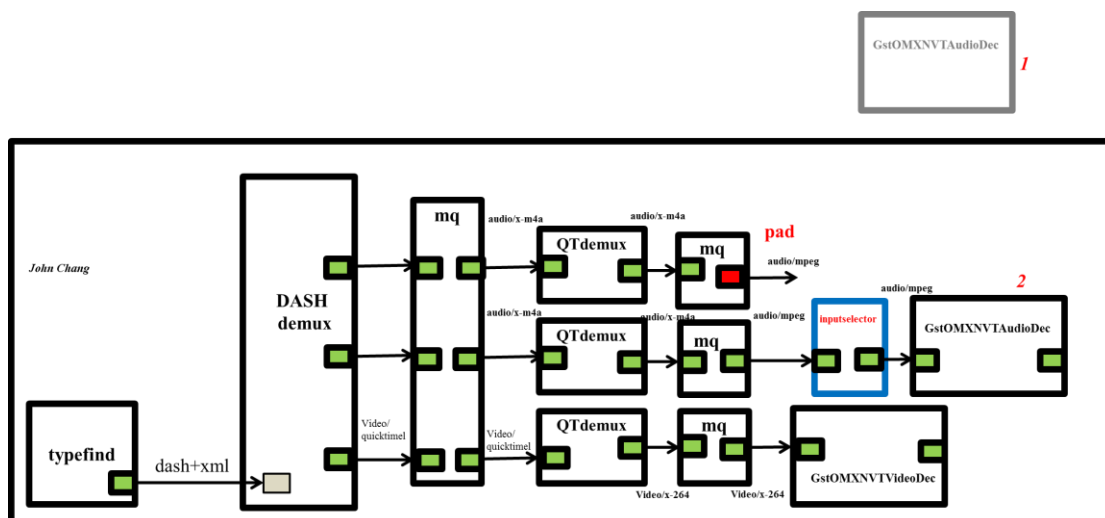


Fig 6 when decode bin is about to add the 2<sup>nd</sup> audio track's decoder

Notice that since the caps is equal to previous one as “audio/mpeg”, the factory will have with the same Gtype as the CODEC instance we intalled at last step. Hence match\_element() will find & return the corresponding node in codec list.

Then we could retrieve the target input selector & connect to it. Finally, set current chain as with deadend by the fact that all consecutive paths have already been built up before.

The steps to handle this case is listed here:

B.1: Get the corresponding input-selector

B.2: Get a new sink pad from input-selector and connect pad to it

B.3: Set chain’s deadend as TRUE to stop the exploration of this chain

Return directly.

Please check the figure 4’s path with prefix = ‘B’.

PS: One may ask why we should set the chain as having deadend?

Here we list the flowchart of gst\_decode\_chain\_is\_complete() below.

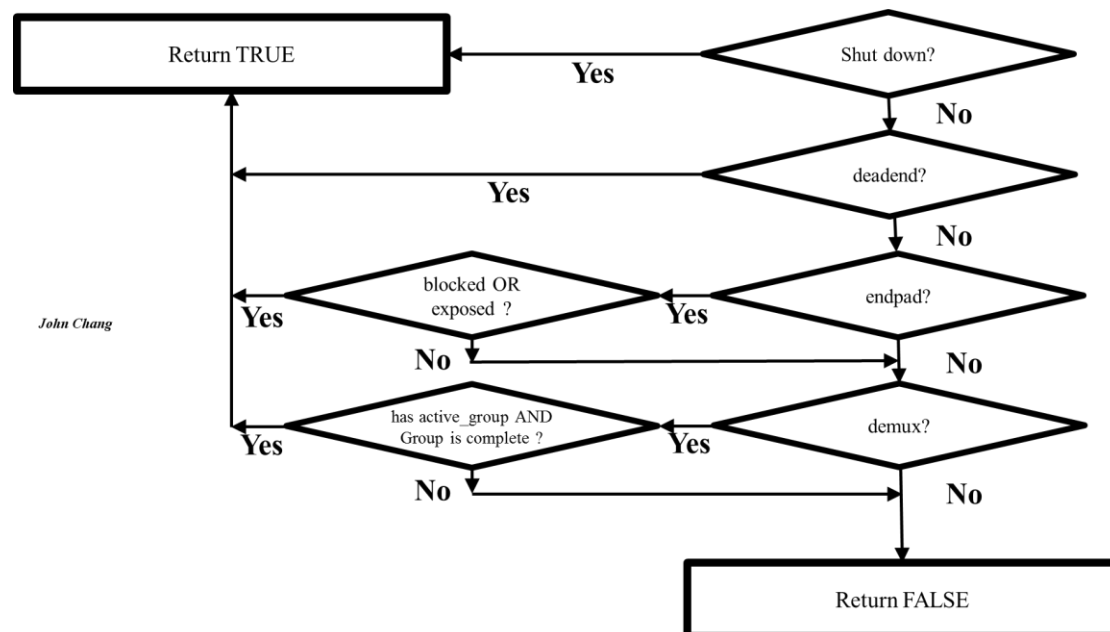


Fig 7 gst\_decode\_chain\_is\_complete()'s flow chart

Please notice that it is possible to notify chain's completion through other ways but it seems that the easiest is by deadend.

It is why we do so but welcome to any suggestion if one has a better idea.

The snapshot it given below to show when we complete adding the second audio CODEC into decode bin.

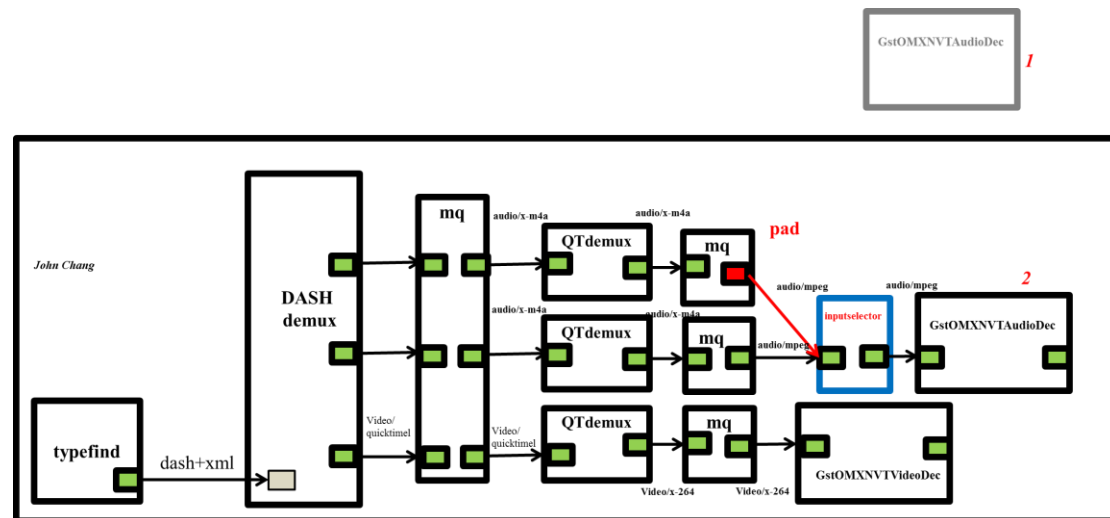


Fig 8 complete adding the second audio CODEC into decode bin

Finally we indicate a tip to which we should pay attention.

For adaptive demux such as DASH, the `pad_added_cb()` callback runs on different threads. For example, in this case it runs on 3 independent threads. So mutex is in need to avoid race condition. It is accomplished by adding `mutex = codec_lock` within decode bin.