Configuration Stores

Hans Verkuil
Cisco Systems Norway

Problem Description

- Android libcamera 3 requires per-frame configurations. So each frame has associated a list of settings that are applied when the frame is captured. In addition it can return per-frame information (histogram, exposure settings, etc.).
- Besides controls the following ioctls can also function as per-frame configuration: S_SELECTION(CROP/COMPOSE), S_INPUT, S_OUTPUT, S_AUDIO, S_AUDOUT, S_PARM (fps), SUBDEV_S_FMT, SUBDEV_S_SELECTION, SUBDEV_S_FRAME_INTERVAL.
- How to solve this without putting undue burden on drivers and applications?

Proposal: use the control framework

- Atomicity is built-in to the control framework. So are all the validation and consistency checks and control events.
- With the new compound type and array support in the control framework it is easy to store structs/arrays as controls.
- You need 'configuration stores' that store all the control values needed for the per-frame configuration. Each store has an ID. ID 0 refers to the current (active) control value, other IDs refer to a specific configuration store.
- When calling QBUF you pass a configuration store ID and the driver will apply the contents of that store when the buffer is made ready for DMA.
- Some experimentation shows two use-cases: update a control every time a configuration store is applied, or update it only if someone set it (fire and forget).

Applying a Configuration Store

- Userspace decides how many config stores are needed, but the driver does set a maximum (probably some multiple of the maximum number of buffers).
- Note that typically only a subset of the controls will have configuration stores. E.g. it makes no sense to have V4L2_CID_POWER_LINE_FREQUENCY in configuration stores.

API Changes

- Add a new flag: V4L2_CTRL_FLAG_HAS_CONFIG_STORE to tell the application that this control supports configuration stores.
- In struct v4l2_buffer the 'reserved2' field is renamed to 'config_store'.

API Changes

- Add a new ioctl: VIDIOC_CONFIG_STORE with argument:

  ```
  struct v4l2_config_store {
          __u32 id;
          __u32 action;
          __u32 flags;
          __u32 reserved[13];
  };
  ```

  where `id` is the configuration store ID and `action` is one of:

  ```
  V4L2_CONF_STORE_OPEN
  V4L2_CONF_STORE_CLOSE
  V4L2_CONF_STORE_APPLY
  ```

  and `flags` is either 0 or V4L2_CONF_STORE_FL_FORGET to signify fire-and-forget.

API Changes

- After `V4L2_CONF_STORE_OPEN` the configuration store ID is stored in struct v4l2_fh. Any ioctl that supports configuration stores (initially only the control ioctls) will be applied to this config store until `V4L2_CONF_STORE_CLOSE`.
- In the future non-control ioctls will be redirected to a control by the core framework if STORE_OPEN is active.