Developing and Using ASCOM Video Drivers

Hristo Pavlov - October 2013

Design Considerations for IVideo

The IVideo interface is designed with the following use cases in mind:

- Instructing the video camera to record a video file and saveing it on the file system
- Occasionally obtaining individual video frames as the camera is running in order to help with focusing, frame rate and image adjustments, field identification and others.
- Provide all video frames in a buffered manner that can be used to do real-time analysis (Optional - if the driver implements it)

Video cameras, unlike standard CCD cameras, produce a constant stream of images that we call video frames. Cameras can typically produce 25 or 30 frames per second (fps) but this number can go up to 200-300 fps. Trying to display every one of those video frames will have a significant impact on the performance of the driver and the hardware. Video is already very resource hungry and implementing a separate streaming in the driver for real time viewing by a client will make the system even more demanding for resources (RAM and CPU). This could affect the integrity of the video record (e.g. result in dropped frames) which is not desired and should be avoided in many applications of video cameras in astronomy. For this reason providing all frames from a video camera in a buffered manner by the driver is an optional part of the ASCOM Video interface.

Still the client will need to see what the camera is pointed to for example to work on the focusing so there is a need for viewing individual images produced by the video camera. This is done is by the client requesting a single individual frame from the camera and doing this as many times as necessary and as fast as the client need and can cope with. If the client is too fast and there is no new image available then the driver will return the same image. The frame id associated with the image should be used by a client application to determine whether the returned frame is a new or an old one.

There are broadly two types of video cameras regarding the way the images are captured by a computer - analogue and digital. Analogue cameras will usually require a video capture card and digital cameras could be connected to the computer via USB, FireWire or a Digital Video port.

Analogue video cameras output frames with a fixed frame rate that cannot be changed while the digital video cameras can have a variable frame rate. When it comes to exposure duration there are again two types of analogue video cameras - integrating and non-integrating.

The integrating video cameras have the shutter open for the duration of the exposure and after the exposure is completed they output the same video frame (the image acquired during the exposure) with their supported frame rate and for the duration of the next exposure.

Non-integrating video cameras have a fixed exposure duration that cannot be changed. Some integrating video cameras also offer exposures shorter than the frame rate interval. In this case the shutter is only open for the corresponding exposure time and the remaining time for up to the frame rate duration the camera is not collecting light. Digital video cameras usually output a single frame for each exposure that they produce but may be also running at a constant frame rate.

The ASCOM Video interface is designed to support all these types of video cameras - analogue, digital, integrating and non-integrating.

Live Streaming, Buffering and Dropped Frames

Video is a real time streaming system. The drivers and clients of ASCOM Video work in a time critical and resource intensive environment and their job is to process large amounts of information while keeping up with the constant number of video frames coming at a real time frame rate. The system is required to be able to handle random delays in the processing of video frames but at the same time having a reasonable tolerance for the maximum delay between when a video frame has become available and when it has been processed. If the delay in the processing of the video frames becomes too big then the system needs to correct this, which is done by skipping one or more frames or dropping them.

The process described above is the normal way any real time video system operates. You may recall instances when you have watched a video on a computer or other device and suddenly the video froze for a few moments because the device became busy. The next thing you see is either the video jumping ahead skipping a part of the video or running at a much faster speed for a little while, until it catches up with the real time streaming.

The drivers that implement ASCOM Video need to do a similar thing. However most likely the API that the driver will be using to obtain video frames will already implement buffering and frame dropping as the Video driver is nothing more than a client to this API and a façade that exposes a standard ASCOM interface to the ASCOM clients. What is more interesting is that video systems that run at a constant frame rate actually don't drop frames but rather duplicate them and this is required so the recorded video has exactly the same frame rate as the source video – always the same number of frames must be found in every second of recorded video. PAL video runs at 25 fps while NTSC runs at almost 30 fps. If in one of those one second PAL intervals the driver which is recording a video file was unable to read 25 frames but for example only read 24, it must still save 25 frames during this second to maintain the frame rate. Because presumably the saving will be much faster operation what the driver can do is save

one of the 24 frames twice and this will result in a frame being duplicated because the a real frame was dropped from the live video stream. We usually say in this case that a frame was dropped but this actually means that the previous frame was repeated twice.

If the driver does all the work required to actually save the individual frames in the video file, then this is how it should handle this situation. Luckily in most cases the underlying API takes care of the saving of the video file automatically and handles dropped/duplicated frames. This is also the reason why the saving of the video file is done by the Video driver and is not left to the ASCOM Video client. This way the ASCOM Video driver can delegate the saving to the underlying framework (e.g. DirectShow) which will be hidden from the client of the driver.

There can be however instances where the ASCOM Video client may want to take care of the saving of the video file. One such example is specialised software that monitors for meteors in real time and only records a short video when a meteor is detected. In order for this to work smoothly the first requirement is the Video driver to support buffering. Only then the client can have guarantees that no frames received by the driver will be dropped. Please note that ASCOM Video drivers only implement the buffering as an optional functionality. If the diver does support buffering then the meteor detection software may also need to implement its own buffer so it keeps frames older than the currently processed frames. Once the software detects a meteor it will likely want to start the recording a couple of second earlier than the frame where the meteor was first detected. This is why a separate buffer is needed. The meteor detection software will then open a new video file and start recording from an older frame using the frames in its own buffer.

If you are working on a Video driver and would like to support buffering you need to consider what mechanism you will be using to drop frames from the buffer once the buffer has been filled up. If the client connects to your diver but doesn't request any video frames you need to make sure that this will not cause an out of memory situation because you keep adding frames to the buffer but no one is consuming them.

TODO: Still lots of things to add …

Acknowledgment

I would like to thank the following people that were involved in the design and adoption of IVideo:

Kyle Goodwin - for his input and assistance in the design of IVideo.

Kyle Goodwin and Evan Warkentine - for testing the interface in the early stages by creating video drivers and video clients.

Peter Simpson and Chris Roland – for their significant work on all the technical details required for publishing IVideo in ASCOM.

Bob Denny and Tim Long – for their support during my journey.

Bob Denny – for his efforts that made this release possible.

Dave Gault – for inspiring me to work on an ASCOM video interface in the first place.

Document Updates

Original Release – October 2013