Some notes on the Sequence Grabber since QuickTime 1.0

```
ComponentResult SGSetChannelClip(SGChannel c, RgnHandle theClip)
```

If the channel has spatial characteristics (i.e. SGGetChannelInfo returns that seqGrabHasBounds) then this call may be used. It puts a clipping region over the bounds specified with SGSetChannelBounds. The default is no clip.

```
ComponentResult SGGetChannelClip(SGChannel c, RgnHandle *theClip)
```

Returns the current clip on the channel. This call is only available if SGSetChannelClip is also available. The caller is responsible for disposing of the returned clip region. If there is no clip, NIL is returned.

```
pascal ComponentResult SGSetDataProc(SeqGrabComponent sg, SGDataProc proc, long refCon)
```

SGSetDataProc allows you to provide a function to the sequence grabber which will be called whenever any SGChannel writes data to the output file. The data proc must be of the following form.

```
typedef pascal OSErr (*SGDataProc)(SGChannel c, Ptr p, long len, long *offset,
long chRefCon, TimeValue time, short writeType, long refCon)
```

The refCon you pass to SGSetDataProc is passed back to you as the last parameter to your function. The channel writing data is passed to you, along with a pointer to the data and its length. Ignore the offset field for now. If you return an error, recording will stop.

The data proc is useful if you want to capture the data being digitized in a form other than in a movie. There is a new flag bit to SGSetDataOutput, seqGrabDontMakeMovie, which tells the sequence grabber that you don't want it to create a movie from the recording. It will still call your data proc, but no data will be written to a file and no movie will be created. This can be useful, for example, if you wanted to send digitized data over the network to another computer. From inside your function, you may need to get the current sample description for the data being written. Call SGGetChannelSampleDescription (described below) to obtain the current sample description.

To remove the data proc, call SGSetDataProc and set the procedure to nil.

The “p” parameter contains a pointer to the data, “len” is the size of that data. If you are creating a movie, you must return the offset that the data was written to in the “offset” field. The contents of the “chRefCon” field depend on the channel. For video channels sample flags, indicating whether it is a key frame or not. It is currently undefined for sound, so it will be zero. The time is the time stamp of the data, in the time scale of the given channel. The “writeType” field indicates the type of write being performed, as explained in the discussion of SGAddMovieData. The “refCon” field is whatever value you passed to “SGSetDataProc”.

```
pascal ComponentResult SGGetChannelSampleDescription(SGChannel c, Handle sampleDesc)
```

SGGetChannelSampleDescription resizes and fills in the provided handle with a sample description for the current data being recorded. It may return an error if called from outside of the recording loop. For video channels it returns an Image Compression Manager ImageDescription. For sound channels, it returns a SoundDescription as defined by the Movie Toolbox.

```
pascal ComponentResult SGGetChannelTimeScale(SGChannel c, TimeScale *scale)
```

SGGetChannelTimeScale returns the TimeScale of the given channel. Typically this is the TimeScale that the Media created by the channel will have. This TimeScale is also the scale on which the TimeValue's passed through the data proc will be expressed in.

```
pascal ComponentResult SGAddMovieData (SeqGrabComponent s, SGChannel c, Ptr p, long len,
            long *offset, long chRefCon, TimeValue time, short writeType)
```

SGAddMovieData is the new preferred way to add channel data to a movie. The channel parameter indicates which channel is adding the data. The Ptr and len fields indicate where the data is, and how large it is. On return the "offset" contains the offset that the data was added to the file at. The "time" field indicates the time stamp for the data. It is expressed in the TimeScale of the given channel. The final field describes what kind of write is taking place.

SGAddMovieData replaces the old combination of calling SGWriteMovieData followed by SGAddFrameReference. It also allows for full support of data procs.

seqGrabWriteAppend - This indicates that the given data should be added to the end of the file, and the offset it was added at returned in the "offset" field.

`seqGrabWriteReserve`- This indicates that the given data should be ignored but that a hole of the given size should be reserved in the data stream for a future write. The offset of the hole should be returned in the "offset" field.

`seqGrabWriteFill` - The given data should be written at the offset indicated by the value pointed to by "offset". This call basically fills in the hole created by the reserve call, although potentially the hole could be filled in pieces.

```
enum {
   seqGrabWriteAppend,
   seqGrabWriteReserve,
   seqGrabWriteFill
};
```

```
pascal ComponentResult SGGetChannelDeviceList (SGChannel c, long selectionFlags,
            SGDeviceList *list)
```

Allocates and returns a list of all the devices which could be connected to the specified SGChannel. The data structure is shown below. A count of the number of items is returned in the structure, together with the index of the currently selected device. The index is zero based.

The only currently defined selection flag is sgDeviceListWithIcons. If you set this flag, the icon field is filled out for all devices. Otherwise it is set to 0.

The names of the devices are returned. Not a Video Digitizer ComponentInstance or Sound Input Driver or whatever.

```
typedef struct SGDeviceName {
      Str63             name;
      Handle            icon;
      long              reserved;          // zero
} SGDeviceName;

typedef struct SGDeviceListRecord {
      short             count;
```

```
        short               selectedIndex;
        long                reserved;           // zero
        SGDeviceName        entry[1];
    } SGDeviceListRecord, *SGDeviceListPtr, **SGDeviceList;

    #define sgDeviceListWithIcons (1)
```

```
pascal ComponentResult SGDisposeDeviceList (SeqGrabComponent s, SGDeviceList list)
```

Disposes of the device list and any icons it contains. Note that this is a call to the sequence grabber itself, not to the channel as is the SGGetDeviceList call.

```
pascal ComponentResult SGAppendDeviceListToMenu (SeqGrabComponent s, SGDeviceList list,
        MenuHandle mh)
```

Takes the given device list and appends it to the provided menu handle. This is useful since often you want to provide the user with a list of devices to choose from.

```
pascal ComponentResult SGSetChannelDevice (SGChannel c, StringPtr name)
```

Given a device name, the SGChannel attempts to open and begin using that device rather than the current one. The name must be one of the names returned by SGGetDeviceList.

Passing the same name as the currently selected device will have no effect.

```
pascal ComponentResult SGSortDeviceList (SeqGrabComponent s, SGDeviceList list)
```

This is a call for SGChannels to make. Applications should never need to make it. The SGChannel can construct its device list in any order that is convenient and then pass it to the sequence grabber's SGSortDeviceList call to have it sorted alphabetically. This call also appropriately updates the "selectedIndex" field of the list.

```
pascal ComponentResult SGNewChannelFromComponent (SeqGrabComponent s, SGChannel *newChanne
        Component sgChannelComponent)
```

SGNewChannelFromComponent lets you create a new SGChannel by specifying the component to create it from, rather than by specifying the channel's subtype (as with SGNewChannel). In all other respects it is the same as SGNewChannel (in fact SGNewChannel now just calls FindNextComponent and then SGNewChannelFromComponent). This call would be useful if you wanted to chose between multiple channel grab components of the same subtype.

```
pascal ComponentResult SGGetIndChannel (SeqGrabComponent s, short index, SGChannel *ref,
        OSType *chanType)
```

This call allows you to index through all the channels current associated with the sequence grabber. The index of the first channel is 1. When there are no more channels, a paramErr is returned. The channel subType is returned as well. Both the "ref" and "chanType" fields may be set to zero if you don't want those values.

```
pascal ComponentResult SGSetSoundInputParameters (SGChannel c, short sampleSize, short
        numChannels, OSType compressionType)
```

SGSetSoundInputParameters is an SGChannel call that is only available for channels whose "seqGrabHasVolume" info flag is set. This call lets you set several parameters relating to sound recording. You can pass the sample size (either 8 or 16), the number of channels (1 or 2) and the compression type ('raw ', 'MAC3', or 'MAC6'). If the current Sound Input Device doesn't support the given settings an error is returned. You may pass zero for any field to imply that it shouldn't be changed. You can pass -1 for any field to imply that it should be reset to its default value.

```
pascal ComponentResult SGGetSoundInputParameters (SGChannel c, short *sampleSize, short
        *numChannels, OSType *compressionType)
```

SGGetSoundInputParameters is an SGChannel call that is only available for channels whose “seqGrabHasVolume” info flag are set. It returns the current settings for the given parameters. You may pass nil for any parameters you aren’t interested in.

```
pascal ComponentResult SGGrabCompressComplete (SGChannel c, Boolean *done, SGCompressInfo
        *ci, TimeRecord *tr)
```

SGGrabCompressComplete is a new video bottleneck routine. It is only called when working with compressed video sources. It combines the functionality of SGGrabComplete, SGComprressFrame, and SGCompressComplete. When it is called, the frame has been grabbed and compressed. The TimeRecord indicates the time at which the frame was grabbed. The SGCompressInfo and TimeRecord are only filled out when “done” is set to true on return.

The prototype for your bottleneck routine is given below.

```
typedef pascal ComponentResult (*GrabCompressCompleteProc)(SGChannel c,
Boolean *done, SGCompressInfo *ci, TimeRecord *tr, long refCon);
```

```
pascal ComponentResult SGDisplayCompress (SGChannel c, Ptr dataPtr, ImageDescriptionHandle
        desc, MatrixRecord *mp, RgnHandle clipRgn)
```

SGDisplayCompress is another new video bottleneck routine for use with compressed video sources. If the compressed data needs to be decompressed for the user to see, SGDisplayCompress is called. If you instead only have an SGDisplayFrame bottleneck installed, the sequence grabber must first decompress the frame for you to transfer using SGDisplayFrame. This can be much less efficient that using SGDisplayCompress which allows you to decompress the frame directly.

The prototype for your bottleneck routine is given below.

```
typedef pascal ComponentResult (*DisplayCompressProc)(SGChannel c, Ptr
dataPtr, ImageDescriptionHandle desc, MatrixRecord *mp, RgnHandle clipRgn,
long refCon);
```

```
pascal ComponentResult SGSetFrameRate (SGChannel c, Fixed frameRate)
```

For video channels only. This call lets you explicitly set a target frame rate for the recorded movie. The frame rate limiting may be performed by the Video Digitizer or by the Sequence Grabber Video Channel, depending on the hardware in use. The frame rate may not be met exactly. In particular, many boards may not be able to deliver high frame rates. You can pass zero for the frame rate to request the default (maximum) frame rate, usually about 30 frames per second.

```
pascal ComponentResult SGGetFrameRate (SGChannel c, Fixed *frameRate)
```

For video channels only. This returns the currently set frame rate for the given channel. By default, this value is set to zero, indicating to grab as fast as possible.

```
pascal ComponentResult SGSetChannelMatrix (SGChannel c, const MatrixRecord *m)
```

This call works only with channels that have spatial bounds. You may set any matrix on the given channel. The matrix maps the current video rectangle (set with SGSetVideoRect) into the destination window. If the matrix cannot be handled by the current device, an error is returned.

Calling SGSetChannelBounds is equivelent to creating a matrix which maps the current video rectangle to the bounds - that is SGSetChannel bounds changes the matrix, much like SetMovieBox changes the movie matrix.

Calling SGSetVideoRect also effects the matrix, as the definition of SGSetVideoRect requires that the adjusted video rectangle continue to appear in the existing destination rectangle.

If the hardware supports it, the matrix can be used to specify flips and rotates.

```
pascal ComponentResult SGGetChannelMatrix (SGChannel c, MatrixRecord *m)
```

This call works only with channels that have spatial bounds. Returns the current display matrix in use by the channel.

```
pascal ComponentResult SGUpdate (SeqGrabComponent s, RgnHandle updateRgn)
```

If your application receives an update event for the window containing a sequence grabber it should call SGUpdate. If you pass nil for the updateRgn, the current visRgn of the window is assumed to be the area that needs updating. You should not call SGUpdate inside a Begin/EndUpdate pair. Rather call it before the BeginUpdate. Your update code should never draw where there is video. This will cause some video digitizers to stop displaying video. You can pass in an updateRgn to SGUpdate to indicate which part of the window was actually changed. For example, in response to an update event you might use the following line, before calling BeginUpdate

```
	SGUpdate(theSG, ((WindowPeek)updateWindow)->updateRgn);
```

```
pascal ComponentResult SGPause (SeqGrabComponent s, Byte paused)
```

The prototype of SGPause has been slightly modified. Rather than passing a Boolean, you pass a Byte. Some new constants have been defined:

```
enum {
	seqGrabUnpause = 0,
	seqGrabPause = 1,
	seqGrabPauseForMenu = 3
};
```

The values for seqGrabUnpause and seqGrabPause mimic the old functionality of SGPause. The new constant, seqGrabPauseForMenu, is only used in preview mode. It indicates that the reason that the sequence grabber is being paused is because you are about to call MenuSelect or PopUpMenuSelect. In this case, the sequence grabber may not pause all channels, in particular sound channel may continue to play. This tends to be less distracting to the user. At some point, video may even continue to play.

```
pascal ComponentResult SGGetPause (SeqGrabComponent s, Boolean *paused)
```

This call tells you whether or not the sequence grabber is currently paused. If a true is a returned, the sequence grabber is paused, otherwise it is not paused. Note that when you call SGStop, the sequence grabber is always returned to an unpaused state.

```
pascal ComponentResult SGGetAlignmentProc (SeqGrabComponent s, AlignmentProcRecordPtr
		alignmentProc)
```

This routine fills in an AlignmentProcRecord which is appropriate for passing to any of the Image Compression Manager alignment routines. By using the routine pointer returned by this function, you can position your sequence grab windows in ideal screen locations for optimal performance.

```
pascal ComponentResult SGSettingsDialog(SeqGrabComponent s, SGChannel c, short numPanels,
		Component *panelList, long flags, SGModalFilterProcPtr proc, long procRefNum)
```

This routine provides a general purpose mechanism for configuring channels associated with the sequence grabber. The sequence grabber implements the dialog using a moveable modal dialog containing a single fixed “monitor” panel managed by the channel itself. There is a second panel which may contain one of a number of sets of settings which are implemented by Sequence Grabber Panel Components. You may pass a list of Panel Components to use to the settings dialog using the numPanels and panelList parameters, or you may set these to zero, and the Sequence Grabber will use all available panels. A long of flags is provided, which should be set to zero for now.

Because the dialog is a moveable modal dialog, you must provide an event filter to process update events for windows in your application. This is done with the proc and procRefCon fields. The format of you event filter proc is shown below:

```
typedef pascal Boolean (*SGModalFilterProcPtr)(DialogPtr theDialog,
EventRecord *theEvent, short *itemHit, long refCon);
```

The only dialog item numbers that are publicly defined are the standard, 1 for OK and 2 for cancel.

If the settings were sucessfully completed, the function returns noErr. If the user canceled the operation, an error of userCanceledError is returned.

Note that the panels are typically allowed to change most parameters of the channel, so you should query the sequence grabber when the call completes to update any stored values you have, such as channel bounds or channel device.

There are currently two standard sub-types of panels defined by Apple. These are sub-types have the same meaning, regardless of the type of channel in use. The sub-types are ‘sour’ which is the panel for choosing the input source and ‘cmpr’ which chooses the compression for the channel.

Note: a discussion of how to write panels for a particular video digitizer, or an application, to deferred to another document.

Other random notes:
Many changes have gone into the video channel component. It now is much smarter about dealing with cards that support only a few digitizing sizes or depths. It now supports cards which do strange analog signal mixing, and is starting to have some support for video digitizers which supply compressed data directly. VD Preflight calls are now used as appropriate, which provides better results with some cards (unfortunately, there are also a few digitizers out there which failed to implement the required Preflight calls and therefore fail with 1.5). A misguided DisposHandle has been fixed in the video prepare code which had some heap trashing abilities. You can now call SGGetBufferInfo while previewing. It may fail (this is particularly true with compressed sources). You can now call SGSetVideoCompressor while recording as long as the depth and compressor fields are set to zero. This allows you to change the quality and key frame rate fields, which may be useful if you want to attempt to influence the data rate. The compress frame complete bottleneck proc is now called.

SGGrabPict has been fixed so that it is actually useful. If the sequence grabber has no channels, SGGrabPict behaves as described in the 1.0 documentation, except that it is probably more reliable now. If the sequence grabber has one or more channels associated with it, a picture is constructed from the current channels using the parameters passed. This makes it very easy to get a picture from the current video source. It is a much safer and more compatible way to obtain a picture, then to try to CopyBits it out of the image buffer.

The sound channel has had a few fixes as well. If it is using hardware play through, it now respects the system volume level, which is consistent with how it handled software simulated play through. It now provides slightly more accurate guesses of the sound input rate in some cases (rounding rather than truncating). It now works correctly with the LC (you can hear sound after you are done recording). You no longer have to call SGStop before setting the sound rate. The volume is no longer lost (it was stored correctly, just plays wrong) after you stop/start the sound channel. When the sound channel is created, (or when you call SGSetSoundInputDriver(channel, 0)) it trys to choose the Sound Input driver associated with the video channel. Sound now correctly records 16 bits samples, and sample rates beyond 32kHz. This means you can actually use the Sequence Grabber to record 44kHz stereo 16 bit sound.

The sequence grabber itself now buffers up writes of data to disk when doing live grabs. This helps increase frame rate somewhat (it also makes the file smaller, since each frame write is no longer aligned to a 512 byte boundary). It does asynchronous disk writes when recording in anticipation of a better day. The sequence grabber now attempts to preallocate the largest free block on your disk for writing to. This was supposed to be in 1.0 (and would have been, but for a “!”). It may help HFS to waste less time.

Both the video and sound channels now fully implement the SetTarget Component Manager message.