

# ASC IIDC/DCAM FireWire Camera API 2.0

## Introduction

The ASC IIDC/DCAM FireWire Camera API allows you to communicate with any IIDC/DCAM compliant FireWire Camera under OS X 10.2 (Jaguar), OS X 10.3 (Panther) and OS X 10.4 (Tiger). The API provides access to the low level camera registers, allows you to access and manipulate camera controls, handles isochronous transmission, collection of data packets and converts the collected packets of data into frames of video for you.

For developers who are new to working with FireWire, the 1394 Trade Association provides basic information and terminology (i.e. isochronous etc.) at <http://www.1394ta.org/Technology/About/TechTalk.htm>. Apple provides Mac specific information on FireWire at <http://developer.apple.com/firewire/overview.html>.

The IIDC specification was developed by the 1394 Trade Association <http://www.1394TA.org>. The specification provides a common guide that camera manufacturers and software developers can follow to allow them communicate with each other over FireWire. This allows any IIDC compliant camera to be used by IIDC compliant software on any operating system. The Full Version 1.3.x IIDC specification is available for purchase from the 1394 Trade Association as document #1999023 entitled "IIDC 1394-based Digital Camera Specification Version 1.30" at <http://www.1394ta.org/Technology/Specifications/specifications.htm>.

## ASC IIDC/DCAM API Implementation

The ASC IIDC FireWire Camera API directly supports the core video features of the 1.04, 1.20, 1.30 and 1.3.1 versions of the IIDC specification, such as:

- Fixed Size Camera support for Formats 0 (up to 640x480 pixels), 1 (up to 1024x768 pixels) and 2 (up to 1200x1600 pixels), Modes 0 to 7 (Monochrome8, YUV411, YUV422, YUV444, RGB24, RGB48 and Monochrome 16) and Frame Rates 0 to 7 (1.875 to 240.00 frames per second), as listed in Table 2 and Table 3 of "Appendix B" at the end of this document.
- Variable (Partial) Sized Camera support for Format 7 and Modes 0 to 7.
- Access to all inquiry, status and control registers for the Camera, such as Brightness, Saturation, Trigger, Temperature or Tilt.
- Access to all Absolute Value CSR inquiry, status and control registers for the Camera, such as Brightness, Shutter, Saturation or Tilt.
- Isochronous channel management and data transfers to extract video data from the camera.
- Access to read or write CSR local and absolute address registers of the Camera for future compatibility with the evolving IIDC specification.

Additionally the ASC IIDC FireWire Camera API provides support for:

- Image conversion (Altivec supported if available) from native YUV411, YUV422, YUV444, Monochrome8, RGB24, Monochrome16 and RGB48 formats to ARGB32 format for display.
- Bayer Video support for 8,12 and 16 bit depths, in four patterns (RGGB, GRBG, GBRG and BGGR) and dithered to ARGB32 or RGB48 space using three selectable algorithms.
- Saving native formats images to disk in bmp, jpeg, pict, photoshop, tiff8, tiff16, png8, png16, fits8 and fits16 file formats.

- Recording video to disk in QuickTime movie formats.
- Camera Selection by Hardware ID.

The ASC IIDC FireWire Camera API does not currently support the following IIDC features directly:

- No Still Image Format 6 support.
- No One Shot Support (however it can be done manually)

However, since the ASC IIDC FireWire Camera API does support reading and writing to any FireWire register address directly, developers are not prevented from accessing or implementing these feature themselves.

## **Camera Controls Registers**

The IIDC FireWire Camera specification defines twenty one camera controls which are listed below and are fully supported by the ASC IIDC FireWire Camera API.

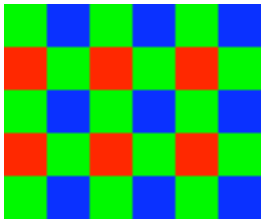
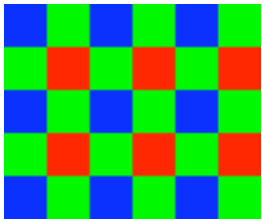
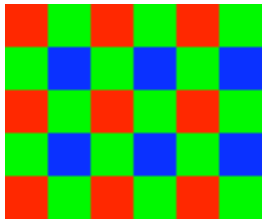
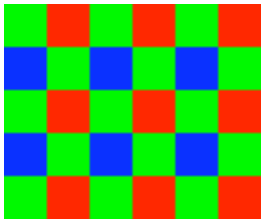
Brightness Control	adjusts the black level of the image
Exposure Control	adjusts the exposure and contrast of the image.
Sharpness Control	adjusts the image sharpness
White Balance Control	adjusts the white balance of color images
Hue Control	adjusts the color phase of images
Saturation Control	adjusts the saturation of color images
Gamma Control	adjusts the gamma setting for color images
Shutter Control	specifies the integration or exposure time for the CCD/CMOS sensor
Gain Control	adjusts the gain circuitry
Iris Control	controls the iris size for the camera optics
Focus Control	controls the focus of the camera optics
Temperature Control	controls the camera internal temperature
Trigger Control	provides support for triggering frame capture, by user specified time or via an external electrical trigger or device.
Trigger Delay	specifies time delays for triggering.
White Shading	controls white shading.
Frame Rate	controls frame rate.
Zoom Control	controls the zoom focal length of the camera optics.
Pan Control	rotates camera field of view along horizontal plane.
Tilt Control	rotates camera field of view along vertical plane.
Optical Filter Control	controls optical filters.

Not all cameras will have implemented all of these controls and none of them are required features for the IIDC specification. For example the controls for iris, focus and zoom have no meaning for a camera that does not have optics. Additionally in some cases the manufacturer of the camera may use a control for another purpose,

so it's best to check the manufacturers specific camera data sheet, especially for something like trigger implementation and support.

### **Bayer Camera Support**

All single CCD/CMOS sensor cameras that deliver color video use some form of Bayer filter array grafted onto the sensor face. A Bayer color filter array (CFA) only passes one specific color (typically red, green, blue or cyan, magenta, yellow, green) to an individual CCD pixel element, similar to what is shown below.

GBRG	BGGR	RGGB	GRBG
			

Most cameras (including still frame Digital Cameras) will extract the raw bayer image, process it internally and then deliver RGB or YUV video frames from it. Although having the camera process the Bayer images is less CPU intensive, it does dramatically (by 2 to 4 times) increase the amount of transmitted data, which reduces frame rate. Also the quality of some camera's bayer to RGB conversion can have problems for scientific, medical or forensic applications, where each raw Pixel of data is considered significant. By receiving the raw Bayer video from the camera to your Mac, you can save bandwidth to run at higher frame rates or use more cameras at the same time, save storage space by only saving the raw frames and post process the raw data into RGB for display without permanently losing any information.

Unfortunately, the "fly in the ointment" is that the IIDC specification does not currently have a universally accepted means of identifying which cameras can deliver Bayer video and in what pattern (i.e. GBRG, BGGR, RGGB or GRBG) it will be.

To work around this IIDC bayer limitation, the ASC IIDC/DCAM FireWire Camera API allows you to treat any 8, 12 or 16 bits per pixel video frame as if it was a frame of Bayer video and can use three different dithering algorithms to create an ARGB32 or RGB48 image from it.

The lowest quality Bayer algorithm needs very little CPU usage and is ideal for previewing the video in real time on slow Macs. The medium quality Bayer algorithm produces better quality images and requires somewhat more CPU usage. The high quality Bayer algorithm produces very good image quality, but is two orders of magnitude slower than the medium quality. The following table summarizes the average time needed to process different size frames of Bayer 8 or 16 bit video into ARGB32/RGB48 images on a G4 933 mHz with 133 mHz bus under OSX 10.2.

Processing Time in milliseconds	Bayer Quality	Input Bayer Image	Output ARG/RGB Image
2.3	Low	640x480 8 bit	640x480 ARGB32
3.3	Medium	640x480 8 bit	640x480 ARGB32
313.4	High	640x480 8 bit	640x480 ARGB32
4.5	Low	640x480 16 bit	640x480 RGB48
6.6	Medium	640x480 16 bit	640x480 RGB48
321.0	High	640x480 16 bit	640x480 RGB48
10.9	Low	1280x960 8 bit	1280x960 ARGB32
14.9	Medium	1280x960 8 bit	1280x960 ARGB32
1252.5	High	1280x960 8 bit	1280x960 ARGB32
23.0	Low	1280x960 16 bit	1280x960 RGB48
29.0	Medium	1280x960 16 bit	1280x960 RGB48
1289.6	High	1280x960 16 bit	1280x960 RGB48

## API Reference

The ASC IIDC/DCAM FireWire Camera API consists of thirty three (33) API calls. The API is documented on a function by function basis and describes:

- Name of the function and necessary parameters to use it.
- Description of the function, it's purpose and what the the expected range of parameters are.
- Usage of the function in a sample code snippet.
- Notes and Conditions for the function, covering likely errors, expected limitations and any known problems or situations to avoid.

OSStatus **ASCDCAMGetListOfCameras**(dcamCameraList \*pTheCameraList);

### Description:

Returns a list of up to sixty three available FireWire DCAM/IIDC cameras in a "dcamCameraList" structure as defined in the "ASC\_DCAM\_API.h" file.

### Usage:

```
dcamCameraList theCameraList;
OSStatus      err;
UInt32        index;
char           theCStr[80];

err = ASCDCAMGetListOfCameras(&theCameraList);
if (err == 0)
{
    if (theCameraList.numCamerasFound > 0)//some cameras exist..
    {
        for (index=0;index<theCameraList.numCamerasFound;index++)
```

```

    {
        if (theCameraList.theCameras[index].isInUse == false)
        {
            //print out a list of cameras that are not in use
            sprintf(theCStr,"Name: %s MFGID: %d UID: %d",
                    theCameraList.theCameras[index].deviceName,
                    theCameraList.theCameras[index].manufacturerID,
                    theCameraList.theCameras[index].uniqueID);
            fprintf(stderr,"%s\n",theCStr);
        }
    }
}
}
}

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

The "deviceName" of the cameras will be reported as "Unknown Model" if the "isInUse" boolean for a camera is true, meaning someone else has exclusive access to the camera.

The state of the "dcamCameraList" structure will only be valid for a relatively short period of time after "ASCDCAMGetListOfCameras()" is called. You must be prepared that at a later time any of the cameras reported may be in use by another application / driver or may be unplugged or powered off and not available.

---

void\* ASCDCAMOpenACamera(UInt32 uniqueID);

### **Description:**

Opens a specific camera by it's unique hardware ID and prepares it for exclusive use. A reference to a block of memory for the camera is returned. All subsequent routines will use the returned camera reference to "talk" to the camera.

### **Usage:**

```

OSStatus      err;
dcamCameraList theCameraList;
OSStatus      err;
UInt32        index;
UInt32        whichCameraID;
void *         theCameraPtr = NULL;

err = ASCDCAMGetListOfCameras(&theCameraList);
if (err == 0)
{
    if (theCameraList.numCamerasFound > 0)//some cameras were found
    {
        index = 0;
        whichCameraID = 0xFFFFFFFF;
        do//loop through the list
        {
            if (theCameraList.theCameras[index].isInUse == false)
            {
                //camera is not in use..
                whichCameraID = theCameraList.theCameras[index].uniqueID;
                //and save it..
            }
        }
    }
}

```

```

    }
    index++;
} while ((index < theCameraList.numCamerasFound) && (whichCameraID == 0xFFFFFFFF));
if (whichCameraID != 0xFFFFFFFF)
{
    //open this camera
    theCameraPtr = ASCDCAMOpenACamera(whichCameraID);
    if (theCameraPtr == NULL)
        err = -1;//failed
}
}
}
}

```

### **Notes and Conditions:**

If the camera you try and open is unavailable (i.e. in use or not attached) or there is another issue such as insufficient memory or other system related problems, then the returned pointer will be NULL.

The "uniqueID" is a physical property of each manufacturer's FireWire camera. Once you have a camera's "uniqueID", you only need to use that parameter to open that camera again.

NEVER dispose of the returned void\* pointer yourself. Always use "ASCDCAMCloseACamera()" to dispose of it, which allows it to clean up anything that has been allocated for the camera as far as buffers and system resources go.

---

OSStatus **ASCDCAMCloseACamera**(void\* pTheCamera);

### **Description:**

Releases the exclusive lock on the camera and completely frees up all system resources and buffers allocated for it. Additionally, if an Isoch Stream is in progress, this will stop the camera from transmitting, shut down the stream, deallocate all Isochronous System resources and dispose of any buffers and threads that have been allocated. It completely cleans up a camera regardless of what it currently is doing.

### **Usage:**

```

OSStatus      err;
err = ASCDCAMCloseACamera(theCameraPtr);

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

---

OSStatus **ASCDCAMSetIODelayTimes**( void\* pTheCamera,  
 UInt32 IOWriteDelayMS, UInt32 IOReadDelayMS);

### **Description:**

Allows you to specify delays in milliseconds between write ("IOWriteDelayMS") or read ("IOReadDelayMS")

transactions to any FireWire addresses, including ones you make directly or indirectly by using the API. Almost all current FireWire cameras do not need this and adding a delay will simply make doing Read/Write IO with the camera that much slower. This should only be necessary for older Sony FireWire Cameras which can require up to 100 millisecond delay after a write to a FireWire address - otherwise those cameras may not work properly.

#### **Usage:**

```
OSStatus          err;
err = ASCDCAMSetIODelayTimes(theCameraPtr,(UInt32) 15,(UInt32) 0);
//have all writes to the camera delayed by 15 milliseconds, giving the camera time to
//complete the operation before proceeding with another read or write request
```

#### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

Please use common sense when specifying the values for "IOWriteDelayMS" or "IOReadDelayMS". Unless your dealing with an old cranky obsolete FireWire camera, you should only need to increase the "IOWriteDelayMS". The default for these paramaters when the Camera is opened using "ASCDCAMOpenACamera()" is zero.

---

```
OSStatus ASCDCAMGetCameraProperties(    void* pTheCamera,
                                       CameraCapabilitiesStruct *cameraProperties);
```

#### **Description:**

Returns information on a camera's general properties, such as camera model name, unique ID, supported formats, supported modes, supported frame rates and which camera control features are implemented. The properties returned in the "CameraCapabilitiesStruct" structure are defined in the "ASC\_DCAM\_API.h" file.

#### **Usage:**

```
short              index;
short              startPoint;
OSStatus           err;
double             frameRateDbl;
CameraCapabilitiesStruct theProperties;

err = ASCDCAMGetCameraProperties(theCameraPtr, &theProperties);
if (err == 0)
{
    //print model name and ID
    fprintf(stderr,"Model:%s ID:%d\n",theProperties.deviceName,theProperties.uniqueID);
    //see if the camera supports format 1 mode 1 (800x600 RGB24 video)
    if (theProperties.format1 == true)//supports format 1
    {
        if (theProperties.format1Modes[1] == true)//supports mode 1
        {
            //print off what frame rates it supports..
            //calculate starting point into the framerate array
            startPoint = 6;//mode0 = 0, mode1 = 6, mode2 = 12, ..,mode5 = 30
            frameRateDbl = 1.875;//lowest frame rate..
        }
    }
}
```

```

    fprintf(stderr,"Supported Frame Rates:\n");
    for (index=startPoint;index < startPoint+6;index++)
    {
        if (theProperties.formatlModesFrameRates[index] == true)
            fprintf(stderr,"%5.4f\n",(float) frameRateDbl);
        frameRateDbl = frameRateDbl*2.00;//1.875, 3.75,7.50, ....,60.00
    }
}
//see if the camera supports brightness control
if (theProperties.hasBright == true)
    fprintf(stderr,"Supports Brightness Control\n");
}

```

### **Notes and Conditions:**

This API call is provided for backwards compatibility for version 1.0 of the ASC IIDC Framework. We recomend you switch to the "ASDCAMGetCameraProperties2()", which provides support for the IIDC 1.3.x specification changes.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

If a property is not supported (i.e. false), then it is (in "theory") not supported by that camera. However, a property that is supported may only be supported in certain Formats or Modes. You need to explicitly test for this condition using the "ASDCAMGetPropertyCapability()" function to determine if it is supported after you change Formats or Modes.

---

```

OSStatus ASCDCAMGetCameraProperties2( void* pTheCamera,
                                     CameraCapabilitiesStruct2 *cameraProperties);

```

### **Description:**

Returns information on a camera's general properties, such as camera model name, unique ID, supported formats, supported modes, supported frame rates and which camera control features are implemented. The properties returned in the "CameraCapabilitiesStruct2" structure are defined in the "ASC\_DCAM\_API.h" file. This new version of the structure supports the absolute CSR register controls and has expanded frame rates to accomodate changes made in the 1.3.x version of the IIDC specification.

### **Usage:**

```

short                index;
short                startPoint;
OSStatus             err;
double               frameRateDbl;
CameraCapabilitiesStruct2 theProperties2;

err = ASCDCAMGetCameraProperties2(theCameraPtr, &theProperties2);
if (err == 0)

```



```

{
    //print model name and ID
    fprintf(stderr,"Model:%s ID:%d\n",theProperties2.deviceName,theProperties2.uniqueID);
    //see if the camera supports format 1 mode 1 (800x600 RGB24 video)
    if (theProperties2.format1 == true)//supports format 1
    {
        if (theProperties2.format1Modes[1] == true)//supports mode 1
        {
            //print off what frame rates it supports..
            //calculate starting point into the framerate array
            startPoint = 8;//mode0 = 0, mode1 = 6, mode2 = 12, ..,mode5 = 30
            frameRateDbl = 1.875;//lowest frame rate..
            fprintf(stderr,"Supported Frame Rates:\n");
            for (index=startPoint;index < startPoint+8;index++)
            {
                if (theProperties2.format1ModesFrameRates[index] == true)
                {
                    fprintf(stderr,"%5.4f\n",(float) frameRateDbl);
                    frameRateDbl = frameRateDbl*2.00;//1.875, 3.75,7.50, ....,240.00
                }
            }
        }
    }
    //see if the camera supports absolute brightness control
    if (theProperties2.hasAbsBright == true)
        fprintf(stderr,"Supports Absolute Brightness Control\n");
}

```

### **Notes and Conditions:**

This is only available in version 2.0 of the IIDC/DCAM Framework.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

If a property is not supported (i.e. false), then it is (in "theory") not supported by that camera. However, a property that is supported may only be supported in certain Formats or Modes. You need to explicitly test for this condition using the "ASDCAMGetPropertyCapability2()" function to determine if it is supported after you change Formats or Modes.

---

```

OSStatus ASCDCAMGetPropertyCapability( void* pTheCamera,
                                       UInt32 cameraPropertySelector,
                                       UInt32 *minVal,
                                       UInt32 *maxVal,
                                       Boolean *canOnePush,
                                       Boolean *canReadOut,
                                       Boolean *canOnOff,
                                       Boolean *canAuto,
                                       Boolean *canManual);

```

### **Description:**

Returns the supported capabilities of a specific camera control.

The "cameraPropertySelector" can be one of the "ASC\_IIDC\_PROPERTY\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which specific control you wish to get information on, such as "ASC\_BRIGHTNESS\_REG" which would return information on the Camera's Brightness control. The following description applies to all controls except for the "ASC\_TRIGGER\_REG", which is handled differently.

The "minVal", "maxVal" and "canManual" values are only relevant if a control can be manually operated. If the "canManual" property is true, then that control supports manual operation. The "minVal" and "maxVal" values are then the allowable range of values that the control can be set to. If the "canManual" is false, then the control does not support manual control and the "minVal" and "maxVal" values have no meaning.

The "canOnePush" value indicates whether this control supports "one push" functionality. A "one push" is a control that is set one time and then the changes remain in effect until set again. An example of such a control could be performing a one time White Balance under specific lighting conditions and then leaving those settings in place for all future uses.

The "canReadOut" value indicates whether this control supports reading of the control's current value. A control may only support automatic adjustment of itself and if "canReadOut" is true, then you may also be able to read the current value out of the control to monitor the changes.

The "canOnOff" value indicates whether the control can be turned on or turned off. If it's false the control is always active, otherwise it can be enabled or disabled.

The "canAuto" value indicates whether the control supports automatic adjustment of itself.

Rather than building separate control functions just for the trigger control ("ASC\_TRIGGER\_REG"), we have modified the returned parameters to support it. Only the "canManual", "canAuto" and "maxVal" have any meaning.

The "canManual" values determine if the Trigger control is implemented.

The "canAuto" value indicates whether the trigger supports polarity switching or not.

The "maxVal" indicates bit wise which trigger modes (modes 0 to 3) are supported and the returned value should be interpreted as shown below.

```
triggermode0 = (Boolean) ((maxVal & 0x08) > 0);  
triggermode1 = (Boolean) ((maxVal & 0x04) > 0);  
triggermode2 = (Boolean) ((maxVal & 0x02) > 0);  
triggermode3 = (Boolean) ((maxVal & 0x01) > 0);
```

If "triggermode0" is true, then Trigger Mode 0 is supported.

Trigger modes 0,1 and 2 all require an external Hardware Trigger Input (i.e. GPIO pin with external cable etc.) and you should consult the manufacturers documentation for signal pulse information and actual usage. Only trigger Modes 2 and 3 depend on current "TRIG\_PARAM\_CNTL" value.

For Trigger Mode 2, the "TRIG\_PARAM\_CNTL" parameter will be 2 or more and indicates the number of integration time units (see manufactures documentation to convert into actual time units) to expose the sensor for. It basically allows you to externally trigger the start of exposure, and the camera exposes the sensor for a programmed amount of time. Please consult your manufacturers documentation for this trigger mode, in case it is used "differently" in actual practice.

For Trigger Mode 3, this parameter will be 1 or more and indicates the number "units" to expose the sensor

for. The number of "units" is defined as the fastest frame rate for the camera, so if the camera's fastest frame rate is 30 fps and a value of 60 is used, then the sensor is exposed for 2 seconds (i.e.  $1/30 * 60 = 2$ ). Please consult your manufacturers documentation for this trigger mode, in case it is used "differently" in actual practice.

### **Usage:**

```
OSStatus                                err;
CameraCapabilitiesStruct                theProperties;
UInt32                                 minVal;
UInt32                                 maxVal,
Boolean                                canOnePush;
Boolean                                canReadOut;
Boolean                                canOnOff;
Boolean                                canAuto;
Boolean                                canManual;
UInt32                                 theValue;

if (theProperties.hasBright == true)//supports the Brightness control
{
    err = ASCDCAMGetPropertyCapability(theCameraPtr,
                                       ASC_BRIGHTNESS_REG,
                                       &minVal,
                                       &maxVal,
                                       &canOnePush,
                                       &canReadOut,
                                       &canOnOff,
                                       &canAuto,
                                       &canManual);

    if (err == 0)//Brightness is also supported in this Format/Mode too
    {
        if (canAuto == true) //enable auto control
        {
            err = ASCDCAMSetPropertyValue(theCameraPtr,
                                           ASC_BRIGHTNESS_REG,
                                           AUTO_CNTL,
                                           1);
        }
        else
        {
            if (canManual == true)
            {
                theValue = (UInt32) ((maxVal + minVal)/2);//set it to the mid point..
                err = ASCDCAMSetPropertyValue(theCameraPtr,
                                               ASC_BRIGHTNESS_REG,
                                               MANUAL_CNTL,
                                               theValue);
            }
        }
    }
}
```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

If the Property is not supported, then you will receive a "-2201" error.

The values returned for a particular control may change depending on the current Format and Mode of the camera. Never assume that the minimum or maximum values are the same after you have changed Format or Modes.

The Trigger control ("ASC\_TRIGGER\_REG") is a special case and needs to be treated differently, as described in the Description: section above.

```
-----  
  
OSStatus ASCDCAMGetPropertyValue( void* pTheCamera,  
                                   UInt32 cameraPropertySelector,  
                                   UInt32 whichControlSelector,  
                                   UInt32 *value);
```

#### **Description:**

Returns the current value for a specified aspect of a control.

The "cameraPropertySelector" can be one of the "ASC\_IIDC\_PROPERTY\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which specific control you wish to get values from, such as reading the current value of the "ASC\_BRIGHTNESS\_REG" control.

The "whichControlSelector" can be one of the "ASC\_IIDC\_CNTL\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which part of a control you wish to get values from, such as "MANUAL\_CNTL" which would return the current value for that control.

You can expect the following values to be returned for these control selectors.

MANUAL_CNTL	returns the current value, between the min. and max. for the control.
AUTO_CNTL	returns 0 (off) or 1 (on)
ON_OFF_CNTL	returns 0 (off) or 1 (on)
ONE_PUSH_CNTL	returns 0 (not in use) or 1 (in use)

For the "ASC\_WHITE\_BAL\_REG" control, you use the following selectors to extra the color balance instead of the "MANUAL\_CNTL" selector. If the Camera is operating in a RGB color mode, then White Balance controls the Blue and Red color intensities. If the Camera is operating in a YUV color mode, then White Balance controls the U or V intensities.

U_OR_B_CNTL	returns the value for the white balance U or Blue control
V_OR_R_CNTL	returns the value for the white balance V or Red control

For the "ASC\_TEMPERATURE\_REG" control, you use the following selectors to extra the temperature information. The temperature values are returned as 10 times the absolute value and must be divided by 10 to convert them to their actual value. This allows the integer temperature to be reported with one decimal place of accuracy. Do not use the "MANUAL\_CNTL" selector.

TARGET_TEMP_CNTL	returns the desired temperature value that the camera is trying to obtain.
CURR_TEMP_CNTL	returns the current camera temperature.

For the "ASC\_TRIGGER\_REG" control, you use the following selectors to extract trigger information.

TRIG_POLARITY_CNTL	returns signal polarity, 0 is Low active input and 1 is High active input.
TRIG_MODE_CNTL	returns the current trigger mode, 0 to 15 are possible, 0 to 3 are supported
TRIG_PARAM_CNTL	returns the optional trigger parameter value for Trigger modes 2 and 3.

Trigger modes 0,1 and 2 all require an external Hardware Trigger Input (i.e. GPIO pin with external cable etc.) and you should consult the manufacturers documentation for signal pulse information and actual usage. Only trigger Modes 2 and 3 depend on current "TRIG\_PARAM\_CNTL" value.

For Trigger Mode 2 , the "TRIG\_PARAM\_CNTL" paramater will be 2 or more and indicates the number of integration time units (see manufactures documentation to convert into actual time units) to expose the sensor for. It basically allows you to externally trigger the start of exposure, and the camera exposes the sensor for a programmed amount of time. Please consult your manufacturers documentation for this trigger mode, in case it is used "differently" in actual practice.

For Trigger Mode 3 , this parameter will be 1 or more and indicates the number "units" to expose the sensor for. The number of "units" is defined as the fastest frame rate for the camera, so if the camera's fastest frame rate is 30 fps and a value of 60 is used, then the sensor is exposed for 2 seconds (i.e.  $1/30 * 60 = 2$ ). Please consult your manufacturers documentation for this trigger mode, in case it is used "differently" in actual practice.

**Usage:**

```
OSStatus                                err;
CameraCapabilitiesStruct                 theProperties;
UInt32                                  minVal;
UInt32                                  maxVal,
Boolean                                 canOnePush;
Boolean                                 canReadOut;
Boolean                                 canOnOff;
Boolean                                 canAuto;
Boolean                                 canManual;
UInt32                                  theValue;
```

[illegible]

```

        fprintf(stderr, "Brightness is %d\n", theValue);
    }
}

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

If the "cameraPropertySelector" or "whichControlSelector" parameters are out of range, or the camera control does not support the selector you are trying to use, it will return "-50" error.

If the control is not supported, then you will receive a "-2201" error.

The values returned for a particular control may change depending on the current Format and Mode of the camera. Never assume that the minimum or maximum values are the same after you have changed Format or Modes.

---

```

OSStatus ASCDCAMSetPropertyValue( void* pTheCamera,
                                UInt32 cameraPropertySelector,
                                UInt32 whichControlSelector,
                                UInt32 value);

```

### **Description:**

Sets the value for a specified aspect of a control.

The "cameraPropertySelector" can be one of the "ASC\_IIDC\_PROPERTY\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which specific control you wish to set values on, such as writing to the "ASC\_BRIGHTNESS\_REG" control.

The "whichControlSelector" can be one of the "ASC\_IIDC\_CNTL\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which part of a control you wish to set values on, such as "MANUAL\_CNTL" which would set the value for that control.

Use the following values with these control selectors. For the "MANUAL\_CNTL", the value must be within the minimum and maximum range of the control as returned by the "ASCDCAMGetPropertyCapability()" function.

MANUAL_CNTL	set to a value that is between the min. and max. for the control.
AUTO_CNTL	set 0 to disable Auto function or 1 to enable Auto function.
ON_OFF_CNTL	set 0 to disable the control or 1 to enable the control.
ONE_PUSH_CNTL	set 1 to perform a one push.

For the "ASC\_WHITE\_BAL\_REG" control, you use the following selectors to extract the color balance instead of the "MANUAL\_CNTL" selector. If the Camera is operating in a RGB color mode, then the White Balance controls the Blue and Red color intensities. If the Camera is operating in a YUV color mode, then the White Balance controls the U or V intensities. The intensities must be within the reported minimum and maximum range of the control, returned by the "ASCDCAMGetPropertyCapability()" function.

U_OR_B_CNTL	set a value for the white balance U or Blue control
V_OR_R_CNTL	set a value for the white balance V or Red control

For the "ASC\_TEMPERATURE\_REG" control, you use the following selector to set a target temperature for the camera. The temperature value must be absolute and is multiplied by 10 times. Do not use the "MANUAL\_CNTL" selector. The target temperature must be within the reported minimum and maximum range of the control, returned by the "ASDCAMGetPropertyCapability()" function.

TARGET_TEMP_CNTL	writes the desired temperature value for the camera.
------------------	--

For the "ASC\_TRIGGER\_REG" control, you use the following selectors to Set trigger information.

TRIG_POLARITY_CNTL	write 0 for Low active input and 1 for High active input.
TRIG_MODE_CNTL	write 0x00 for mode0, 0x01 for mode1, 0x02 for mode2 and 0x03 for mode3
TRIG_PARAM_CNTL	sets an optional trigger parameter value for Trigger modes 2 and 3.

Trigger modes 0,1 and 2 all require an external Hardware Trigger Input (i.e. GPIO pin with external cable etc.) and you should consult the manufacturers documentation for signal pulse information and actual usage. Only trigger Modes 2 and 3 depend on current "TRIG\_PARAM\_CNTL" value.

For Trigger Mode 2 , the "TRIG\_PARAM\_CNTL" paramater will be 2 or more and indicates the number of integration time units (see manufactures documentation to convert into actual time units) to expose the sensor for. It basically allows you to externally trigger the start of exposure, and the camera exposes the sensor for a programmed amount of time. Please consult your manufacturers documentation for this trigger mode, in case it is used "differently" in actual practice.

For Trigger Mode 3 , this paramater will be 1 or more and indicates the number "units" to expose the sensor for. The number of "units" is defined as the fastest frame rate for the camera, so if the camera's fastest frame rate is 30 fps and a value of 60 is used, then the sensor is exposed for 2 seconds (i.e.  $1/30 * 60 = 2$ ). Please consult your manufacturers documentation for this trigger mode, in case it is used "differently" in actual practice.

### Usage:

```

OSStatus          err;
CameraCapabilitiesStruct theProperties;
UInt32            minVal;
UInt32            maxVal,
Boolean           canOnePush;
Boolean           canReadOut;
Boolean           canOnOff;
Boolean           canAuto;
Boolean           canManual;
UInt32            theValue;

err = ASCDCAMGetPropertyCapability(theCameraPtr,
                                   ASC_EXPOSURE_REG,
                                   &minVal,
                                   &maxVal,
                                   &canOnePush,
                                   &canReadOut,
                                   &canOnOff,
                                   &canAuto,
                                   &canManual);

if (err == 0)//Brightness is also supported in this Format/Mode too
{

```

```

if (canAuto == true) //enable auto control
{
    err = ASCDCAMSetPropertyValue(theCameraPtr,
                                ASC_EXPOSURE_REG,
                                AUTO_CNTL,
                                1);
}
else
{
    if (canManual == true)//set the manual mode if supported..
    {
        theValue = (UInt32) ((maxVal + minVal)/2);//set it to the mid point..
        err = ASCDCAMSetPropertyValue(theCameraPtr,
                                    ASC_EXPOSURE_REG,
                                    MANUAL_CNTL,
                                    theValue);
    }
}
}

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

If a control was previously in auto mode and you wish to change it to manual mode, you must first turn off auto mode before you try setting the control value. Being in auto mode will prevent the manual settings from "sticking".

You can not use "normal" controls and absolute controls (see "ASCDCAMSetABSPropertyValue()" below) at the same time to set a control register. If you use "ON\_OFF\_ABS\_CNTL" to turn on absolute controls for a control register, then changes to that control register made using "ASCDCAMSetPropertyValue()" will be ignored. So to avoid confusion it's best to use one method, either absolute or normal and stick with it.

The values returned for a particular control may change depending on the current Format and Mode of the camera. Never assume that the minimum or maximum values are the same after you have changed Format or Modes.

---

```

OSStatus ASCDCAMGetABSPropertyCapability(    void* pTheCamera,
                                             UInt32 cameraPropertySelector,
                                             float *minVal,
                                             float *maxVal,
                                             Boolean *canReadOut,
                                             Boolean *canManual);

```

### **Description:**

Returns the supported absolute register capabilities of a specific camera control. IIDC cameras that follow the 1.30 or higher specification may optionally support absolute floating point values for camera controls in "real world" units. For example a camera that support absolute values for shutter control will return CCD



integration times in floating point seconds, where as Gain is in dB decibal values, White Balance is in degrees Kelvin, Tilt and Pan are in degrees of rotation, Saturation and White Balance are percentages.

The "cameraPropertySelector" can be one of the "ASC\_IIDC\_PROPERTY\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which specific control you wish to get information on, such as "ASC\_BRIGHTNESS\_REG" which would return information on the Camera's Brightness control.

The "minVal", "maxVal" and "canManual" values are only relevant if a control can be manually operated. If the "canManual" property is true, then that control supports manual operation. The "minVal" and "maxVal" values are then the allowable range of values that the control can be set to. If the "canManual" is false, then the control does not support manual control and the "minVal" and "maxVal" values have no meaning.

The "canReadOut" value indicates whether this control supports reading of the control's current value. A control may only support automatic adjustment of itself and if "canReadOut" is true, then you may also be able to read the current value out of the control to monitor the changes.

### **Usage:**

```
OSStatus          err;
CameraCapabilitiesStruct theProperties;
float             minVal;
float             maxVal;
Boolean           canReadOut;
Boolean           canManual;
float             theValue;

if (theProperties.hasAbsShutter == true)//supports the absolute Shutter control
{
    //get the minimum and maximum exposure time in seconds..
    err = ASCDCAMGetABSPROPERTYCapability(theCameraPtr,
                                           ASC_SHUTTER_REG,
                                           &minVal,
                                           &maxVal,
                                           &canReadOut,
                                           &canManual);
}
```

### **Notes and Conditions:**

Only available in version 2.0 of the IIDC/DCAM Framework.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

If the Property is not supported, then you will receive a "-2201" error.

The values returned for a particular control may change depending on the current Format and Mode of the camera. Never assume that the minimum or maximum values are the same after you have changed Format or Modes.

The Trigger control ("ASC\_TRIGGER\_REG") is a special case and needs to be treated differently, as described in



```

        &theValue);
    if (err == 0)
        fprintf(stderr, "Brightness is %f\n", theValue);
    }
}
}

```

### **Notes and Conditions:**

Only available in version 2.0 of the IIDC/DCAM Framework.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

If the "cameraPropertySelector" or "whichControlSelector" parameters are out of range, or the camera control does not support the selector you are trying to use, it will return "-50" error.

If the control is not supported, then you will receive a "-2201" error.

The values returned for a particular control may change depending on the current Format and Mode of the camera. Never assume that the minimum or maximum values are the same after you have changed Format or Modes.

---

```

OSStatus ASCDCAMSetABSPropertyValue(    void* pTheCamera,
                                         UInt32 cameraPropertySelector,
                                         UInt32 whichControlSelector,
                                         float value);

```

### **Description:**

Sets the absolute register value for a specified aspect of a control. IIDC cameras that follow the 1.30 or higher specification may optionally support absolute floating point values for camera controls in "real world" units. For example a camera that support absolute values for shutter control will return CCD integration times in floating point seconds, where as Gain is in dB decibal values, White Balance is in degrees Kelvin, Tilt and Pan are in degrees of rotation, Saturation and White Balance are percentages.

The "cameraPropertySelector" can be one of the "ASC\_IIDC\_PROPERTY\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which specific control you wish to set values on, such as writing to the "ASC\_BRIGHTNESS\_REG" control.

The "whichControlSelector" can be one of the "ASC\_IIDC\_CNTL\_SELECTORS" values as listed in the "ASC\_DCAM\_API.h". It determines which part of a control you wish to set values on, such as "MANUAL\_CNTL" which would set the value for that control.

The "whichControlSelector" can be either "MANUAL\_CNTL" or "ON\_OFF\_ABS\_CNTL". It determines which part of a control you wish to set values to.

MANUAL_CNTL	set to a value that is between the min. and max. for the control.
ON_OFF_ABS_CNTL	set 0 to disable absolute controls or 1 to enable absolute control.

### **Usage:**

```
OSStatus          err;
CameraCapabilitiesStruct theProperties;
float             minVal;
float             maxVal,
Boolean          canReadOut;
Boolean          canManual;
float             theValue;

err = ASCDCAMGetABSPROPERTYCapability(theCameraPtr,
                                     ASC_SHUTTER_REG,
                                     &minVal,
                                     &maxVal,
                                     &canReadOut,
                                     &canManual);

if (err == 0) //Shutter is also supported in this Format/Mode too
{
    if (canManual == true) //set the manual mode if supported..
    {
        //first we need to enable absolute controls BEFORE we can set them this way..
        //this also will disable all calls to ASCDCAMSetPropertyValue()
        err = ASCDCAMSetABSPROPERTYValue(pTheCamera,
                                         ASC_SHUTTER_REG,
                                         ON_OFF_ABS_CNTL,
                                         (float) 1.00);

        if (err == 0)
        {
            //okay now we can change the shutter speed to a floating point time in seconds..
            theValue = (float) ((maxVal + minVal) * 0.50); //set it to the mid point..

            err = ASCDCAMSetABSPROPERTYValue(pTheCamera,
                                             ASC_SHUTTER_REG,
                                             MANUAL_CNTL,
                                             theValue);

            //Now turn it off so that we can goi back to using ASCDCAMSetPropertyValue()
            ASCDCAMSetABSPROPERTYValue(pTheCamera,
                                       ASC_SHUTTER_REG,
                                       ON_OFF_ABS_CNTL,
                                       (float) 0.00);
        }
    }
}
```

### **Notes and Conditions:**

Only available in version 2.0 of the IIDC/DCAM Framework.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

You can not use absolute controls and "normal" controls at the same time to change a control register. If you use "ON\_OFF\_ABS\_CNTL" to turn on absolute controls for a control register, then changes to that control register

made using "ASCDCAMSetPropertyValue()" will be ignored. So to avoid confusion it's best to use one method, either absolute or normal and stick with it.

The values returned for a particular control may change depending on the current Format and Mode of the camera. Never assume that the minimum or maximum values are the same after you have changed Format or Modes.

---

```
OSStatus ASCDCAMGetFMRProperties( void* pTheCamera,
                                  UInt32 format,
                                  UInt32 mode,
                                  double frameRate,
                                  UInt32 *maxWidth,
                                  UInt32 *maxHeight,
                                  UInt32 *widthInc,
                                  UInt32 *heightInc,
                                  UInt32 *leftInc,
                                  UInt32 *topInc,
                                  UInt32 *maxPacketSize,
                                  UInt32 *packetSizeInc,
                                  UInt32 *supportedColorCoding);
```

#### **Description:**

Given a specific Format, Mode and frame Rate, this function returns all the camera imaging parameters by reading them from the Cameras Registers.

The "format" must be 0,1,2 or 7.

The "mode" must be in the range of 0 to 7.

The "frameRate" is only applicable for formats 0,1 or 2 and should be in the range of 1.875 to 240.00. It is ignored for format 7.

The "maxWidth" parameter is the maximum width in pixels that the camera supports.

The "maxHeight" parameter is the maximum height in pixels that the camera supports.

The "widthInc" parameter is the pixel increment that the width of the camera's imaging area can be adjusted by. If it has the same value as the "maxWidth", then the width is fixed and can not be changed. If it is less than the "maxWidth" value, then the width of the imaging area can be adjusted in those increments. For example, if "widthInc" is 256, and the camera's "maxWidth" is 1024, then camera's imaging width can be set to 256, 512, 768 or 1024 pixels.

The "heightInc" parameter is the pixel increments that the height of the camera's imaging area can be adjusted by. If it has the same value as the "maxHeight", then the height is fixed and can not be changed. If it is less than the "maxHeight" value, then the height of the imaging area can be adjusted in those increments. For example, if "heightInc" is 300, and the camera's "maxHeight" is 900, then the camera's imaging height can be set to 300, 600 or 900 pixels.

The "leftInc" parameter is the pixel increments that camera's imaging area can be offset on the horizontal axis. If it has the same value as the "maxWidth", then the left coordinate can not be changed. If it is less than the "maxWidth" value, then the left edge of the imaging area can be adjusted in those increments, same as

"widthInc" can.

The "topInc" parameter is the the pixel increments that camera's imaging are can be offset on the vertical axis. If it has the same value as the "maxHeight", then the top coordinate can not be changed. If it is less than the "maxHeight" value, then the top edge of the imaging area can be adjusted in those increments, same as "heightInc" can.

The "maxPacketSize" parameter is the maximum size of a FireWire packet in bytes that can be supported, which can be used to specify band width usage.

The "packetSizeInc" parameter is the the packet size increment (in bytes) that camera's packet size can be adjusted. If it has the same value as the "maxPacketSize", then the packet size can not be adjusted. If it is less than the "maxPacketSize" value, then the packet size can be adjusted in those increments. For example, if "packetSizeInc" is 1024 and "maxPacketSize" is 3096, then the packet size can be set to 1024, 2048 or 3096 bytes.

The "supportedColorCoding" parameter indicates what image formats that the camera may support. For formats 0,1 or 2 this will be a single format. For format 7, multiple formats may be supported. To determine what formats are supported use the bit wise tests as shown in the usage section below.

### Usage:

```
OSStatus      err;
UInt32        theFormat;
UInt32        theMode;
double        theFrameRateDbl;
UInt32        maxWidth;
UInt32        maxHeight;
UInt32        widthInc;
UInt32        heightInc;
UInt32        leftInc;
UInt32        topInc;
UInt32        maxPacketSize;
UInt32        packetSizeInc;
UInt32        supportedColorCoding;
Boolean        hasMono8,hasYUV411,hasYUV422,hasYUV444,hasRGB24,hasMono16,hasRGB48;
Boolean        hasBayer8,hasBayer16;
theFormat      = 0;//format 0
theMode        = 0;//mode 0 so this is 160x120 YUV444 video..
theFrameRateDbl = 60.00; //as fast as possible..
err = ASCDCAMGetFMRProperties(theCameraPtr,
                             theFormat,theMode,theFrameRateDbl,
                             &maxWidth,&maxHeight,
                             &widthInc,&heightInc,
                             &leftInc,&topInc,
                             &maxPacketSize,&packetSizeInc,
                             &supportedColorCoding);

if (err == 0)
{
    //now determine what color modes are supported ..
    //bit wise test.. for Format 0, Mode 0 it should only be hasYUV444 == true.
    hasMono8 = (Boolean) (((supportedColorCoding & (0x80000000 >> 0))>> (31-0)) == 1);
    hasYUV411= (Boolean) (((supportedColorCoding & (0x80000000 >> 1))>> (31-1)) == 1);
    hasYUV422= (Boolean) (((supportedColorCoding & (0x80000000 >> 2))>> (31-2)) == 1);
    hasYUV444= (Boolean) (((supportedColorCoding & (0x80000000 >> 3))>> (31-3)) == 1);
    hasRGB24 = (Boolean) (((supportedColorCoding & (0x80000000 >> 4))>> (31-4)) == 1);
    hasMono16= (Boolean) (((supportedColorCoding & (0x80000000 >> 5))>> (31-5)) == 1);
}
```

```

hasRGB48 = (Boolean) (((supportedColorCoding & (0x80000000 >> 6))>> (31-6)) == 1);
//Bayer cameras may set these bits for IIDC 1.3.1 specification cameras.
hasBayer8 =(Boolean) (((supportedColorCoding & (0x80000000 >> 9))>> (31-9)) == 1);
hasBayer16 =(Boolean) (((supportedColorCoding & (0x80000000 >>10))>> (31-10)) == 1);
}

```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

If the "format" or the "mode" are not supported, a "-2201" error will be returned. You can determine which formats and modes are supported using "ASDCAMGetCameraProperties()".

The frame rate parameter does not apply to format 7. For formats 0, 1 or 2, if the frame rate you specify can not be accommodated (i.e. too high or low), the closest one available will automatically be selected for you. No error will be reported in this case. You can use the "ASDCAMGetFMRLTWHCB()" function to determine what rate it was actually set to.

---

```

OSStatus ASCDCAMSetFMR(      void* pTheCamera,
                             UInt32 format,
                             UInt32 mode,
                             double frameRate);

```

### **Description:**

Configures the Fire Wire camera to use a specified, format, mode and frame rate.

The "format" must be 0, 1 or 2 and not all cameras support all formats. For format 7 use the "ASDCAMSet7MR()" function below.

The "mode" must be in the range of 0 to 7. Not all cameras support all modes.

The "frameRate" should be in the range of 1.875 to 240.00. This value controls the size of each firewire packet and the faster the frame rate the larger the packet size and more FireWire bandwidth that is used. The more band width that is used the fewer cameras that can be used at the same time.

### **Usage:**

```

OSStatus          err;
UInt32            theFormat;
UInt32            theMode;
double            theFrameRateDb1;
CameraCapabilitiesStruct theProperties;

theFormat          = 1; //format 1
theMode            = 4; //mode 4 so 1024x768 RGB24 video
theFrameRateDb1    = 0.00; //as slow as possible..

err = ASCDCAMGetCameraProperties(theCameraPtr, &theProperties);

```

```

if (err == 0)
{
    //see if the camera supports format 1 mode 4
    if (theProperties.format1 == true)//supports format 1
    {
        if (theProperties.format1Modes[theMode] == true)//supports mode 4
        {
            //yes, so configure the camera to transmit 1024x768 RGB24 video
            err = ASCDCAMSetFMR(theCameraPtr,theFormat,theMode,theFrameRateDbl);
        }
    }
}

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

If the specified format or mode parameter are not supported, a "-50" value will be returned. You can determine which formats and modes are supported using "ASCDCAMGetCameraProperties()".

If the frame rate you specify can not be accommodated (i.e. too high or low), the closest available frame rate will automatically be selected for you. No error will be reported in this case. You can use the "ASCDCAMGetFMRLTWHCB()" function to determine what rate it was actually set to.

If the camera is already transmitting isochronous data when you call "ASCDCAMSetFMR()", the API will automatically stop the isochronous transfer of video, make your supplied format, mode and frame rate changes to the camera and then restarts the isochronous transfer for you. If it returns a non zero error response in this situation, the Camera may no longer be transmitting video after it returns and it's best to stop the Isoch stream using the appropriate (i.e. ASCDCAMAllocateReleaseIsochChannel()) API commands to get everything into a known state.

---

```

OSSStatus ASCDCAMSet7MR(    void* pTheCamera,
                           UInt32 mode,
                           UInt32 left,
                           UInt32 top,
                           UInt32 width,
                           UInt32 height,
                           UInt32 colorCodingID,
                           UInt32 bytesPerPacket);

```

### **Description:**

Configures the Fire Wire camera for transmitting Format 7 Video via the supplied parameters.

The "mode" parameter must be in the range of 0 to 7. Not all cameras support all modes.

The "left", "top", "width" and "height" parameters allow you to specify what portion of the CCD array to extract images from and define your Region Of Interest (ROI). Not all cameras support adjusting all of these parameters.



The "bytesPerPacket" parameter specifies how many bytes are to be sent in each FireWire packet, which controls both bandwidth used and frame rate. The larger the bytes per packet, the more video is delivered and the faster the frame rate. However the more bytes you send per packet, the less bandwidth other cameras have and the fewer number of cameras that can be transmitting at the same time.

The "colorCodingID" indicates what video formats to deliver and should be one of the "ASC\_FORMAT7\_COLOUR\_ID" types listed in "ASC\_DCAM\_API.h".

### **Usage:**

```
CameraCapabilitiesStruct    theProperties;
UInt32                    mode;
UInt32                    left;
UInt32                    top;
UInt32                    width;
UInt32                    height;
UInt32                    colorCodingID;
UInt32                    bytesPerPacket;

mode = 2;
left = 0; //no left offset
top = 0; //no top offset
width = 0; //use camera maximum width
height = 0; //use camera maximum height
bytesPerPacket = 0; //use camera maximum packet sizes

//we ASSUME that this camera supports kMONO8_ID, you should check first
colorCodingID = kMONO8_ID; //defined in "ASC_FORMAT7_COLOUR_ID" in "ASC_DCAM_API.h"

err = ASCDCAMGetCameraProperties(theCameraPtr, &theProperties);
if (err == 0)
{
    //see if the camera supports format 7 mode 2
    if (theProperties.format7 == true) //supports format 7
    {
        if (theProperties.format7Modes[mode] == true) //supports mode 2
        { //set it..
            err = ASCDCAMSet7MR(theCameraPtr,
                                mode,
                                left,
                                top,
                                width,
                                height,
                                colorCodingID,
                                bytesPerPacket);
        }
    }
}
```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

If Format 7 or the modes parameter are not supported, a "-50" value will be returned. You can determine which formats and modes are supported using "ASDCAMGetCameraProperties()" described above.

If you supply values of 0 for the "width" or "height" or "bytesPerPacket", then the "ASDCAMSet7MR()" function will automatically fill them in using the maximum values that the camera supports. If the "width" parameter is changed, then the "left" parameter will also be set to 0. If the "height" parameter is changed, then the "top" parameter will also be set to 0.

You should verify your settings against the values returned by "ASDCAMGetFMRProperties()" for proper range and what features are supported. If a parameter is unsupported or out of range you will receive a "-50" error.

Your combined "width" and "left" parameters can not exceed the maximum width of the CCD array (i.e. width + left <= max width) or a "-50" error will be returned.

Your combined "height" and "top" parameters can not exceed the maximum height of the CCD array (i.e. height + top <= max height) or a "-50" error will be returned.

If the camera is already transmitting isochronous data when you call "ASDCAMSet7MR()", the API will automatically stop the isochronous transfer of video, make your format, mode and frame rate changes to the camera, and then restarts the isochronous transfer for you. If it returns a non zero error response in this situation, the Camera may no longer be transmitting video after it returns and it's best to stop the Isoch stream using the appropriate "ASDCAMAllocateReleaseIsochChannel()" API commands to get everything into a known state.

---

```
OSStatus ASCDCAMGetFMRLTWHCB( void* pTheCamera,
                               UInt32 *format,
                               UInt32 *mode,
                               double *frameRate,
                               UInt32 *left,
                               UInt32 *top,
                               UInt32 *width,
                               UInt32 *height,
                               OSType *cCCCType,
                               UInt32 *bitsPerPixel);
```

### **Description:**

Returns the FireWire cameras current configuration settings, which are stored internally and not read from the Camera Registers (i.e. no latency).

The "format", "mode" and "frameRate" parameters indicate what the camera was previously been programmed for. For Format 7, the "frameRate" parameter will have no meaning and can be ignored.

The "left", "top", "width" and "height" parameters indicate what the Region of Interest (ROI) and dimensions of the camera imaging area is set to. The "left", "top" will only be applicable for format 7 mode and can be ignored otherwise.

The "cCCCType" indicates what pixel format the video is and will follow Apple's QuickTime "Four CC" names of "k444YpCbCr8CodecType", "k422YpCbCr8CodecType", "kYUV411PixelFormat" and "kRawCodecType". The

"kRawCodecType" indicates the format is either an RGB or monochrome image format.

The "bitsPerPixel" value indicates the number of bits in each pixel and is also used to determine the format of the "kRawCodecType" video, as shown below.

"kRawCodecType"	with 8 bits per pixel is Monochrome8
"kRawCodecType"	with 24 bits per pixel is RGB24
"kRawCodecType"	with 16 bits per pixel is Monochrome16
"kRawCodecType"	with 48 bits per pixel is RGB48

### **Usage:**

```
OSStatus err;
```

```
err = ASCDCAMSetFMR(theCameraPtr,0,0,240.00); //160x120 YUV444 at up to 60 fps..
if (err==0)
{
    err = ASCDCAMGetFMRLTWHCB( theCameraPtr,
                                &format, //0
                                &mode, //0
                                &frameRateDbl, //should be 240.00, could be less
                                &left, //0
                                &top, //0
                                &width, //160
                                &height, //120
                                &cCCType, // k444YpCbCr8CodecType
                                &bitsPerPixel); // 24bpp
}
```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

You must have called either "ASCDCAMSetFMR" or "ASCDCAMSet7MR" functions prior to using this call or you will get a "-50" error.

---

```
OSStatus ASCDCAMGetQuadlet( void* pTheCamera,
                             UInt16 addressHi,
                             UInt32 addressLo,
                             Boolean isAbsoluteAddress,
                             UInt32 *quadletValue);
```

### **Description:**

Reads the "quadletValue" value at a specified address. A "quadlet" is a 4 byte sized value, such as a long integer, unsigned long integer or float.

The "isAbsoluteAddress" controls how the read is to be performed. If it's "false" then the address is a relative offset from the Cameras Base Command Register Address. If it's "true", then the address is an absolute 48 bit value.

The "addressHi" and "addressLo" are the address values you wish to read at. The address space that FireWire can access is expressed as 64 bit value, with the upper 16 bits being for node information and the lower 48 bits being the address. For relative offsets, only the "addressLo" needs to be filled in.

#### **Usage:**

```
OSStatus      err;
UInt32        addressHi;
UInt16        addressLo;
UInt32        theValue;
```

```
addressLo = 0x058C;//read this relative address so only addressLo is relevant
err = ASCDCAMGetQuadlet(theCameraPtr,0,addressLo,false,&theValue);
```

```
addressHi = 0xFFFF;//upper 16 bits
addressLo = 0xF00C3480;//lower 32 bit address
err = ASCDCAMGetQuadlet(theCameraPtr,addressHi,addressLo,true,&theValue);
```

#### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

This should only be used for Cameras that have custom interfaces and need to access the registers directly.

---

```
OSStatus ASCDCAMSetQuadlet( void* pTheCamera,
                             UInt16 addressHi,
                             UInt32 addressLo,
                             Boolean isAbsoluteAddress,
                             UInt32 quadletValue);
```

#### **Description:**

Writes the "quadletValue" 4 byte value to a specified address. A "quadlet" is a 4 byte sized value, such as a long integer or unsigned long integer or float.

The "isAbsoluteAddress" controls how the write is to be performed. If it's "false" then the address is a relative offset from the Cameras Base Command Register Address. If it's "true", then the the address is an absolute 48 bit value.

The "addressHi" and "addressLo" are the address values you wish to write to. The address space that FireWire can access is expressed as a 64 bit value, with the upper 16 bits being for node information and the lower 48 bits being the address. For relative offsets, only the "addressLo" needs to be filled in.

#### **Usage:**

```
OSStatus      err;
UInt32        addressHi;
UInt16        addressLo;
UInt32        theValue;
```

```
addressLo = 0x0588;//write to this relative address
```

```

theValue = 0xFEDCBA98;
err = ASCDCAMSetQuadlet(theCameraPtr, 0,addressLo,false,theValue);

addressHi = 0xFFFF;//upper 16 bits
addressLo = 0xF00C3488;//lower 32 bit address
theValue = 0x00000001;//value
err = ASCDCAMSetQuadlet(theCameraPtr, addressHi,addressLo,true,theValue);

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

This should only be used for cameras that have custom interfaces and need to access the registers directly. Be especially careful with changing registers that control basic camera functions, such as isoch transmission, as the Camera Code Object and the Physical Camera can get of synch if you change cached values behind it's back - resulting in unpredictable behavior.

---

```

OSStatus ASCDCAMAllocateReleaseIsochChannel( void* pTheCamera,
                                              Boolean allocatIsoch);

```

### **Description:**

Controls the allocation and destruction of system resources and internal buffers, which are used to transmit frames of video via isochronous transfer.

### **Usage:**

```

OSStatus      err;

err = ASCDCAMSetFMR(theCameraPtr,1,4,(double)3.750);//set to format 1 mode 4 and 3.75 fps
if (err == 0)
{
    err = ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,true);//start the isoch stream
    if (err == 0)
    {
        //Start the camera transmitting video..
        err = ASCDCAMStartStopVideo(theCameraPtr,true);
        if (err != 0)//failed stop the stream..
            err = ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,false);
    }
}
}

```

### **Notes and Conditions:**

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

Before calling "ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,true)" to start isochronous transmission, you must have called either "ASCDCAMSetFMR()" or "ASCDCAMSet7MR()" to configure the camera for transmission. If you don't you will receive a "-50" error.

Calling "ASDCAMAllocateReleaseIsochChannel(theCameraPtr,false)" will automatically call "ASDCAMResumePauseVideo()" if necessary to stop the camera from transmitting video.

---

```
OSStatus ASCDCAMStartStopVideo( void* pTheCamera,
                                Boolean startVideo);
```

**Description:**

Controls the starting and stopping of the transfer of isochronous data packets between the camera and the Mac.

**Usage:**

```
OSStatus err;
```

```
err = ASCDCAMSetFMR(theCameraPtr,1,4,(double)3.750); //set to format 1 mode 4 and 3.75 fps
if (err == 0)
{ //start the isoch stream
    err = ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,true);
    if (err == 0)
    { //Start the camera transmitting video..
        err = ASCDCAMStartStopVideo(theCameraPtr,true);
        if (err != 0) //failed stop the stream..
            err = ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,false);
    }
}
```

**Notes and Conditions:**

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of other error codes.

Upon starting the camera, the default is to immediately begin collecting isochronous data packets into frames of video.

---

```
OSStatus ASCDCAMResumePauseVideo( void* pTheCamera,
                                   Boolean resumeVideo);
```

**Description:**

The "ASDCAMResumePauseVideo()" function allows to eliminate the CPU usage of packet collection and video processing, but without incurring the delay in starting or stopping the camera as happens with "ASDCAMStartStopVideo()". This is primarily for developers that need to completely free up CPU usage for image processing or analysis after capturing a frame.

If "resumeVideo" is "false", then isochronous data packets will not be collected or processed. If "resumeVideo" is "true", then frames of video data will be collected. The time latency of going from "resumeVideo = false" to "resumeVideo = true" is within one frame.

## OSStat

### Notes and Conditions:

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

If you use `"ASDCAMStartStopVideo()"` to start or stop the camera, then the `"resumeVideo"` state will be reset to match the state of the camera. This basically means you need to call `"ASDCAMResumePauseVideo()"` to reset it to your desired state after calling `"ASDCAMStartStopVideo()"`.

```
OSStatus ASCDCAMConvertRawFrameToARGB32( void* pTheCamera,
void* srcPtr,
UInt32 width,
```

```

UInt32 height,
OSType cCCCType,
UInt32 bitsPerPixel,
void *destPtr,
UInt32 rowByteWidth);

```

### **Description:**

Converts a frame of unprocessed video data into a ARGB32 buffer, suitable for saving to disk from a buffer or displaying to screen in a PixMapHandle, GWorldPtr or WindowRef.

The "srcPtr" buffer contains a raw frame of video data extracted from the camera. You are responsible for the allocation and destruction of this buffer.

The "width", "height", "cCCCType" and "bitsPerPixel" parameters indicates the size and format of the image data in the "srcPtr". You should be extracting these parameters using the "ASCDCAMGetFMRLTWHCB()" function after you change the format or mode of the camera.

For cameras that deliver Bayer video data, the "cCCCType" should be changed to one of the appropriate Bayer "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h" so for decoding into the ARGB32. The ASCDCAM API supports 8, 12 and 16 bit Bayer data formats, in four (RGGB, GRBG, GBRG and BGGR) Bayer mosaic patterns and converts to ARGB using one of three (low, medium and high) quality dithering algorithms.

The "destPtr" is a buffer large enough to hold the resulting ARGB32 image. It must have been allocated with at least "width\*height\*4" bytes of space. You are responsible for the allocation and destruction of this buffer.

The "rowByteWidth" parameter allows you to specify the number of bytes in a line of video in the "destPtr" destination buffer, which accounts for any optimized padding at the end of each line in the buffers. This allow the conversion routine to place the the ARGB32 pixels directly into a PixMapHandle based Mac structures such as GWorldPtrs, CGrafPtrs and WindowRefs. If your using a simple buffer as a destination, then use a value of 0. For AltiVec routines, you'll want to ensure the end of each line is padded to a multiple of 16 bytes.

### **Usage:**

```

OSStatus err;
UInt32 format;
UInt32 mode;
double frameRateDbl;
UInt32 left;
UInt32 top;
UInt32 width;
UInt32 height;
OSType cCCCType;
UInt32 bitsPerPixel;
Ptr baseAddress;
UInt32 pixRowBytes;
Ptr theRawFramePtr;
PixMapHandle hPixMap;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB( theCameraPtr,
                           &format,
                           &mode,
                           &frameRateDbl,
                           &left,
                           &top,

```





## **Description:**

Creates an appropriate QuickTime type (i.e. via "QTNewGWorld()") GWorldPtr from a frame of unprocessed video data usable by Quicktime for various functions (i.e. decompressing in to windows). The image in the GWorldPtr is delivered in full bit depth, so if the frame contains > 8 bits per pixel components (i.e. Monochrome 16 or RGB48), the GWorld will be at that depth.

The "srcPtr" buffer contains a raw frame of video data extracted from the camera. You are responsible for the allocation and destruction of this buffer.

The "width", "height", "cCCCType" and "bitsPerPixel" parameters indicates the size and format of the image data in the "srcPtr". You should be extracting these parameters using the "ASCDCAMGetFMRLTWHCB()" function after you change the format or mode of the camera.

For cameras that deliver Bayer video data, the "cCCCType" should be changed to one of the appropriate Bayer "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h" so for decoding into the ARGB32. The ASCDCAM API supports 8, 12 and 16 bit Bayer data formats, in four (RGGB, GRBG, GBRG and BGGR) Bayer mosaic patterns and converts to ARGB using one of three (low, medium and high) quality dithering algorithms.

The "theGWorld" is a reference to a GWorldPtr. Upon successfully completing the conversion, the GWorldPtr will be created for you and contains the image data in an appropriate format. You are responsible for Disposing of this GWorldPtr using "DisposeGWorld()" when you are done using it.

## **Usage:**

```
OSStatus  err;
UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
Ptr        theRawFramePtr;
UInt32    theGWBytesPerPixels;
OSType    theGWFormat;
GWorldPtr  theGWorld;
PixMapHandle  thePixMap;
ImageDescriptionHandle theImDesHndl;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &width,
                              &height,
                              &cCCCType,
                              &bitsPerPixel);

if (err==0)
{
    theRawFramePtr = NewPtrClear((long) ((width*height*bitsPerPixel)/8L));
    err = MemError();
}
```

```

if (err==0)
{
    err = ASCDCAMGrabRawFrame(    theCameraPtr,
                                  theRawFramePtr,
                                  2000 );//wait up to 2 seconds
    if ((err==0) && (theRawFramePtr != NULL))
    {
        err = ASCDCAMConvertRawFrameToWorld( theCameraPtr,
                                              theRawFramePtr,
                                              width,
                                              height,
                                              cCCCType,
                                              bitsPerPixel,
                                              &theGWorld);

        if ((err==0) && (theGWorld != NULL))
        {
            //see what format the GWorld is in..
            thePixMap = GetGWorldPixMap(theGWorld);
            err = MakeImageDescriptionForPixMap(thePixMap,&theImDesHndl);
            if ((err==0) && (theImDesHndl != NULL))
            {
                short theIMDepth;
                theGWFormat = (*(theImDesHndl)).cType;
                theIMDepth = (*(theImDesHndl)).depth;
                DisposeHandle((Handle) theImDesHndl);//clean up..
                switch (theGWFormat)
                {
                    case kRawCodecType://Mono8, RGB24 or ARGB32
                        if (theIMDepth == k32ARGBPixelFormat)
                            theGWBytesPerPixel = 4;//ARGB32
                        else
                            if (theIMDepth == k24RGBPixelFormat)
                                theGWBytesPerPixel = 3;//RGB24
                            else
                                theGWBytesPerPixel = 1; //Mono 8
                        break;
                    case k16GrayCodecType:
                        theGWBytesPerPixel = 2;//Mono16
                        break;
                    case k48RGBCodecType:
                        theGWBytesPerPixel = 6;//RGB48
                        break;
                }
            }
            DisposeGworld(theGWorld);//clean up..
        }
        DisposePtr(theRawFramePtr);//clean up..
    }
}
}

```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error

codes. You do not need to dispose of the GWorld if the routine fails, it will take care of that itself and will set the GWorldPtr to NULL on return.

Unless the returned GWorld is of a "k32ARGBPixelFormat" or "k8IndexedGrayPixelFormat" types, you can not simply use a QuickDraw call "CopyBits()" to transfer the image into a window for display or manipulation. QuickDraw does not understand GWorlds that are RGB24, RGB48 or Mono16 and will not operate properly on them. For these other depths formats, you must use some of the QuickTime routines such as "FDecompressImage()" or "DecompressSequenceFrame()" or write your own blitter.

If the GWorldPtr returned is "k8IndexedGrayPixelFormat" type, then be forewarned that the values in the GWorld will be inverted (i.e. 255 - value). For historical reasons Apple has defined the color black (RGB 0x000000) to have an indexed value of "255" and "white" (RGB 0xFFFFFF) to have an index value of "0", which is the reverse of how it should be. To be consistent with Apple's inverted CLUT, the API inverts the values in the GWorld PixMap for you so that they are correctly displayed.

---

```
OSStatus ASCDCAMConvertRaw16FrameToRGB48( void* pTheCamera,
                                           void* srcPtr,
                                           UInt32 width,
                                           UInt32 height,
                                           OSType cCCCType,
                                           UInt32 bitsPerPixel,
                                           void* destPtr
                                           UInt32 rowByteWidth);
```

### **Description:**

Converts a frame of Bayer12 or 16 bit video data into an existing RGB48 buffer, suitable for processing or saving to disk from a buffer.

The "srcPtr" buffer contains a raw frame of video data extracted from the camera. You are responsible for the allocation and destruction of this buffer.

The "width", "height", "cCCCType" and "bitsPerPixel" parameters indicates the size and bayer format of the image data in the "srcPtr". You should be extracting these parameters using the "ASCDCAMGetFMRLTWHCB()" function after you change the format or mode of the camera.

The "cCCCType" should be changed to one of the appropriate Bayer "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h" so for decoding into the ARGB32. The ASCDCAM API supports 12 and 16 bit Bayer data formats, in four (RGGB, GRBG, GBRG and BGGR) Bayer mosaic patterns and converts to ARGB using one of three (low, medium and high) quality dithering algorithms.

The "destPtr" is a buffer large enough to hold the resulting RGB48 image. It must have been allocated with at least "width\*height\*12" bytes of space. You are responsible for the allocation and destruction of this buffer.

The "rowByteWidth" parameter allows you to specify the number of bytes in a line of video in the "destPtr" destination buffer, which accounts for any optimized padding at the end of each line in the buffers. This allows the conversion routine to place the the RGB48 pixels directly into a PixMapHandle based Mac structures such as GWorldPtrs, CGrafPtrs and WindowRefs. If you're using a simple buffer as a destination, then use a value of 0. For AltiVec routines, you'll want to ensure the end of each line is padded to a multiple of 16 bytes and all

lines start on a 16 byte boundary.

Usage:

[illegible]

```

        UnlockPixels(hPixMap);
    }
}
DisposePtr(theRawFramePtr)
}
}

```

### **Notes and Conditions:**

This API call is only available in version 2.0 of the IIDC/DCAM Framework.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

---

```

OSStatus ASCDCAMGrabRawFrame(    void* pTheCamera,
                                void * bufferPtr,
                                UInt32 timeOutDurationMS );

```

### **Description:**

Extract a Raw Frame of video from the camera.

The "bufferPtr" is a buffer large enough to hold one frame of video. Use the "ASDCAMGetFMRLTWHCB()" function to extract the width, height and bits per pixel parameters needed to calculate the minimum size of the buffer.

The "timeOutDurationMS" is the time in milliseconds to wait for a frame to arrive. This should be at least as long as the current frame rate. Use a value of 0xFFFFFFFF for no timeout.

### **Usage:**

```

OSStatus  err;
UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
Ptr       theRawFramePtr;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &width,

```

```

        &height,
        &cCCCType,
        &bitsPerPixel);

if (err==0)
{
    theRawFramePtr = NewPtrClear((long) ((width*height*bitsPerPixel)/8L));
    err = MemError();
    if (err==0)
    {
        err = ASCDCAMGrabRawFrame(    theCameraPtr,
                                      theRawFramePtr,
                                      5000 );//5 second timeout

        myDoSomethingWithRawFrame(theRawFramePtr);
        DisposePtr(theRawFramePtr)
    }
}
}

```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You must have called "ASCDCAMAllocateReleaseIsochChannel()" and "ASCDCAMStartStopVideo()" to start video being transmitted or you will receive a "-50" error.

If the time to capture a frame exceeds the specified "timeOutDurationMS" value, it will return a "-9074" error.

---

```

OSStatus ASCDCAMGrabARGB32Frame(    void* pTheCamera,
                                    void *destARGB32Ptr,
                                    OSType displayFormat,
                                    UInt32 timeOutDurationMS);

```

### **Description:**

Extract a Raw Frame of video from the camera, convert it to ARGB32 and return it in the supplied buffer. This routine will allocate buffers as needed for the conversion process.

The "destARGB32Ptr" is a buffer large enough to hold one frame of ARGB32 bit video. Use the "ASCDCAMGetFMRLTWHCB()" function to extract the width and height of the frame and then use them to calculate the minimum bytes needed in the buffer (width \* height \* 4).

Unless the Camera delivers Bayer format video, the "displayFormat" should always be "kASC\_ARGB" as defined in the "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h". The "ASCDCAMGrabARGB32Frame()" will handle any conversion as necessary.

If the cameras does deliver Bayer video data, the "displayFormat" should be changed to one of the appropriate Bayer "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h" for decoding into ARGB32. The ASCDCAM API supports 8, 12 and 16 bit Bayer data formats, in four (RGGB, GRBG, GBRG and BGGR) Bayer

mosaic patterns and converts to ARGB using one of three (low, medium and high) quality dithering algorithms.

The "timeOutDurationMS" is the time in milliseconds to wait for a frame to arrive. At a minimum, this should be twice as long as the current frame rate is. Use a value of 0xFFFFFFFF for no timeout, however the routine may never complete in this case and your applicaiton could hang.

#### **Usage:**

```
OSStatus  err;
UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
Ptr       theARGB32Ptr;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &width,
                              &height,
                              &cCCCType,
                              &bitsPerPixel);

if (err==0)
{
    //allocate ARGB32 buffer
    theARGB32Ptr = NewPtrClear((long) (width*height*4));
    err = MemError();
    if (err==0)
    {
        err = ASCDCAMGrabARGB32Frame( theCameraPtr,
                                      theARGB32Ptr,
                                      1000); //1 second timeout
        myDoSomethingWithARGB32Frame(theARGB32Ptr);
        DisposePtr(theARGB32Ptr)
    }
}
```

#### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

Additional buffers will be allocated during the conversion process, so it is possible a -108 (memFullErr) may be returned if insufficient memory is available.

You must have called "ASCDCAMAllocateReleaseIsochChannel()" and "ASCDCAMStartStopVideo()" to start video being transmitted or you will receive a "-50" error.



If the time to capture a frame exceeds the specified "timeOutDurationMS" value, it will return a "-9074" error.

---

```
OSStatus ASCDCAMSaveFrameToDisk( void* pTheCamera,
                                  FSSpec fileLocation,
                                  OSType fileFormat,
                                  OSType displayFormat,
                                  UInt32 timeOutDurationMS);
```

### **Description:**

Extracts a frame of video from the camera and saves it to a specified location on disk in a specified file format. This routine will allocate buffers as needed for the conversion process.

The "fileLocation" is an FSSpec structure indicating where the file will be saved to. You can programatically create your own FSSpec's or used Navigation Services to get one base don the users choice. Existing files will not be over written.

The "fileFormat" determines what format to save the file in. Currently supported types available are :

kQTFileTypePhotoShop	8 bit ARGB Photoshop ".psd" file format
kQTFileTypeJPEG	8 bit ARGB JPEG format - loss less quality
kQTFileTypeBMP	8 bit ARGB BMP format
kQTFileTypePicture	8 bit ARGB PICT format
kQTFileTypePNG	8 or 16 bit ARGB PNG
kQTFileTypeTIFF	8 or 16 bit ARGB TIFF
kASCFileTypeFITS	8 or 16 bit NASA FITS format - only for Monochrome images.

The bit depth (i.e. RGB48 or Monochrome16) that the camera is in determines the bit depth used for PNG, TIFF and FITS files. If the camera is delivering RGB48 bit images, then the resulting TIFF or PNG file will be in 16 bits per pixel and if it is delivering a YUV, RGB24 or Monochrome8 format, the image will be 8 bits per pixel.

Unless the Camera delivers Bayer format video, the "displayFormat" should always be "kASC\_ARGB" as defined in the "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h". The "ASCDCAMSaveFrameToDisk()" will handle any internal conversion as necessary.

If the cameras does deliver Bayer video data, the "displayFormat" should be changed to one of the appropriate Bayer "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h" for decoding into ARGB space. The ASCDCAM API supports 8, 12 and 16 bit Bayer data formats, in four (RGGB, GRBG, GBRG and BGGR) Bayer mosaic patterns and converts to ARGB using one of three (low, medium and high) quality dithering algorithms.

The "timeOutDurationMS" is the time in milliseconds to wait for a frame to arrive. This should be at least as long as the current frame rate. Use a value of 0xFFFFFFFF for no timeout, however the routine may never complete in this case and your applicaiton could hang.

### **Usage:**

```
FSSpec    fileLocation,
OSType    fileFormat,
OSType    displayFormat,
OSStatus  err;
```

```

UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
Ptr       theARGB32Ptr;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &width,
                              &height,
                              &cCCCType,
                              &bitsPerPixel);

if (err==0)
{
    //use the returned setting to figure out what formats we want to export..
    displayFormat = kASC_ARGB;//this camera does not do Bayer, so use ARGB..
    if (cCCCType == 'kRawCodecType')//RGB or Monochrome
    {
        if ((bitsPerPixel == 8) || (bitsPerPixel == 16)) //monochrome
            fileFormat = kASCFileTypeFITS;//save it as a FITS file..
        else //save as ARGB TIFF
            fileFormat = kQTFileTypeTIFF;
    }
    else//YUV formats..
        fileFormat = kQTFileTypePicture;//save as a pict..

    err = ASCDCAMSaveFrameToDisk(    theCameraPtr,
                                    fileLocation,//your file location
                                    fileFormat,
                                    displayFormat,
                                    2000);//wait 2 ms

}

```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

You must have called "ASCDCAMAllocateReleaseIsochChannel()" and "ASCDCAMStartStopVideo()" to start video being transmitted or you will receive a "-50" error.

Additional buffers will be allocated during the conversion process, so it is possible a -108 (memFullErr) may be returned if insufficient memory is available.

If the time to capture a frame exceeds the specified "timeOutDurationMS" value, it will return a "-9074" error.

If you specify "kASCFileTypeFITS" as the file format type and the image you are trying to save is not monochrome 8 or 16 bit, you will receive a "-50" error.

---

```
OSStatus ASCDCAMSaveGWorldtoDisk( void* pTheCamera,
                                   GWorldPtr theGWorld,
                                   FSSpec fileLocation,
                                   OSType fileFormat);
```

### **Description:**

Given a frame of RGB video contained in a GWorldPtr, this functions saves it to a specified location on disk in a specified file format. This routine will allocate buffers as needed for the conversion process.

The "fileLocation" is an FSSpec structure indicating where the file will be saved to. You can programatically create your own FSSpec's or used Navigation Services to get one based on the users choice. Existing files will not be over written.

The "fileFormat" determines what format to save the file in. Currently supported types available are :

kQTFileTypePhotoShop	8 bit ARGB Photoshop ".psd" file format
kQTFileTypeJPEG	8 bit ARGB JPEG format - loss less quality
kQTFileTypeBMP	8 bit ARGB BMP format
kQTFileTypePicture	8 bit ARGB PICT format
kQTFileTypePNG	8 or 16 bit ARGB PNG
kQTFileTypeTIFF	8 or 16 bit RGB TIFF
kASCFileTypeFITS	8 or 16 bit NASA FITS format - only for Monochrome images.

The bit depth (i.e. ARGB 32, RGB48, Monochrome16 etc.) that the camera is in determines the bit depth used for PNG, TIFF and FITS files. If the camera is delivering RGB48 bit images, then the resulting TIFF or PNG file will be in 16 bits per pixel component and if it is delivering a YUV, RGB24 or Monochrome8 format, the image will be 8 bits per pixel component.

### **Usage:**

```
FSSpec    fileLocation
OSType    fileFormat;
OSType    displayFormat;
OSStatus  err;
UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
GWorldPtr theGWorldPtr = NULL;
void*     theBufferPtr = NULL;
```

```

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &width,
                              &height,
                              &cCCCType,
                              &bitsPerPixel);

if (err==0)
{
    //allocate our frame buffer
    theBufferPtr = malloc((width*height*bitsPerPixel)/8);
    if (theBufferPtr == NULL)
        err = -108;//a memFullErr so we are out of ram..
    else //sucess, grab a frame and 0xFFFFFFFF means wait forever for it to complete..
        err = ASCDCAMGrabRawFrame(theCameraPtr,theBufferPtr,0xFFFFFFFF);

    if (err==0)//convert raw frame data into an appropriate RGB GWorld..
        err = ASCDCAMConvertRawFrameToGWorld(    theCameraPtr
                                                  theBufferPtr,
                                                  width,
                                                  height,
                                                  cCCCType,
                                                  bitsPerPixel
                                                  &theGWorldPtr);

    //clean up the frame buffer..
    if (theBufferPtr != NULL)
        free(theBufferPtr);
}
if (err==0)
{
    //use the returned setting to figure out what formats we want to export..
    displayFormat = kASC_ARGB;//this camera does not do Bayer, so we request ARGB..
    if (cCCCType == kRawCodecType)//RGB or Monochrome
    {
        if ((bitsPerPixel == 8) || (bitsPerPixel == 16)) //monochrome
            fileFormat = kASCFileTypeFITS;//save it as a Mono FITS file..
        else //save as ARGB TIFF
            fileFormat = kQTFileTypeTIFF;
    }
    else//YUV formats..
        fileFormat = kQTFileTypePicture;//save as a pict..

    err = ASCDCAMSaveGWorldtoDisk(    theCameraPtr,
                                     theGWorldPtr,//the GworldPtr to save out
                                     fileLocation,//your file location
                                     fileFormat,//file format
                                     displayFormat);

    if (theGWorldPtr != NULL)
        DisposeGWorld(theGWorldPtr);
}

```

## **Notes and Conditions:**

This API call is only available in version 2.0 of the IIDC/DCAM Framework.

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

Additional buffers will be allocated during the conversion process, so it is possible a -108 (memFullErr) may be returned if insufficient memory is available.

You must have called "ASDCAMAllocateReleaseIsochChannel()" and "ASDCAMStartStopVideo()" to start video being transmitted or you will receive a "-50" error.

If you specify "kASCFileTypeFITS" as the file format type and the image you are trying to save is not monochrome 8 or 16 bit, you will receive a "-50" error.

```
-----  
  
OSStatus ASCDCAMSaveRAWFrameToDisk(    void* pTheCamera,  
                                       void * bufferPtr,  
                                       UInt32 width,  
                                       UInt32 height,  
                                       UInt32 bitsPerPixel,  
                                       FSSpec fileLocation,  
                                       OSType fileFormat,  
                                       OSType displayFormat);
```

## **Description:**

Saves a RAW frame of video extracted from the camera to a specified location on disk in a specified file format. This routine will allocate buffers as needed for the conversion process - therefore it is not thread safe.

The "fileLocation" is an FSSpec structure indicating where the file will be saved to. You can programatically create your own FSSpec's or used Navigation Services to get one based on the users choice. Existing files will not be over written.

The "fileFormat" determines what format to save the file in. Currently supported types available are :

kQTFileTypePhotoShop	8 bit ARGB Photoshop ".psd" file format
kQTFileTypeJPEG	8 bit ARGB JPEG format - loss less quality
kQTFileTypeBMP	8 bit ARGB BMP format
kQTFileTypePicture	8 bit ARGB PICT format
kQTFileTypePNG	8 or 16 bit ARGB PNG
kQTFileTypeTIFF	8 or 16 bit ARGB TIFF
kASCFileTypeFITS	8 or 16 bit NASA FITS format - only for Monochrome images.

The bit depth (i.e. RGB48 or Monochrome16) that the camera is in determines the bit depth used for PNG, TIFF and FITS files. If the camera is delivering RGB48 bit images, then the resulting TIFF or PNG file will be in 16 bits per pixel and if it is delivering a YUV, RGB24 or Monochrome8 format, the image will be 8 bits per pixel.

Unless the Camera delivers Bayer format video, the "displayFormat" should always be "kASC\_ARGB" as defined in the "ASC\_DISPLAY\_FORMATS" types listed in the "ASC\_DCAM\_API.h". The

"ASDCAMSaveRAWFrameToDisk()" will handle any internal conversion as necessary.

If the camera does deliver Bayer video data, the "displayFormat" should be changed to one of the appropriate Bayer "ASC\_DISPLAY\_FORMATS" types like "kMQ\_ASCBAYER\_RGGB" listed in the "ASC\_DCAM\_API.h" for decoding into ARGB space. The ASCDCAM API supports 8, 12 and 16 bit Bayer data formats, in four (RGGB, GRBG, GBRG and BGGR) Bayer mosaic patterns and converts to ARGB using one of three (low, medium and high) quality dithering algorithms.

### Usage:

```
FSSpec    fileLocation,
OSType    fileFormat,
OSType    displayFormat,
OSStatus  err;
UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
void*     theBufferPtr = NULL;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &width,
                              &height,
                              &cCCCType,
                              &bitsPerPixel);

if (err==0)
{
    //allocate our frame buffer
    theBufferPtr = malloc((width*height*bitsPerPixel)/8);
    if (theBufferPtr == NULL)
        err = -108;//a memFullErr so we are out of ram..
    else //sucess, grab a frame and 0xFFFFFFFF means wait forever for it to complete..
        err = ASCDCAMGrabRawFrame(theCameraPtr,theBufferPtr,0xFFFFFFFF);
}

if (err==0)
{
    //use the returned setting to figure out what formats we want to export..
    displayFormat = kASC_ARGB;//this camera does not do Bayer, so use ARGB..
    if (cCCCType == 'kRawCodecType')//RGB or Monochrome
    {
        if ((bitsPerPixel == 8) || (bitsPerPixel == 16)) //monochrome
            fileFormat = kASCFileTypeFITS;//save it as a FITS file..
        else //save as ARGB TIFF
            fileFormat = kQTFileTypeTIFF;
    }
}
```

```

else//YUV formats..
    fileFormat = kQTFileTypePicture;//save as a pict..
err = ASCDCAMSaveRAWFrameToDisk( theCameraPtr,
                                theBufferPtr,
                                width,
                                height,
                                bitsPerPixel,
                                fileLocation,//your file location
                                fileFormat,//file format..
                                displayFormat);//display format..
}
//clean up the frame buffer if it was allocated..
if (theBufferPtr != NULL)
    free(theBufferPtr);

```

### **Notes and Conditions:**

This API call is only available in version 2.0 of the IIDC/DCAM Framework.

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

Additional buffers will be allocated during the conversion process, so it is possible a -108 (memFullErr) may be returned if insufficient memory is available.

You must have called "ASCDCAMAllocateReleaseIsochChannel()" and "ASCDCAMStartStopVideo()" to start video being transmitted or you will receive a "-50" error.

If you specify "kASCFileTypeFITS" as the file format type and the image you are trying to save is not monochrome 8 or 16 bit, you will receive a "-50" error.

---

```

OSStatus ASCDCAMCancelGrabFrame(void* pTheCamera);

```

### **Description:**

Allows you to cancel a pending "ASCDCAMSaveFrameToDisk()", "ASCDCAMGrabARGB32Frame()" or "ASCDCAMGrabRawFrame()" call before it completes. This allows you to cancel a frame capture prior to it either timing out or completing on it's own, and is primarily intended for use with Cameras doing long exposures. It will take up to 1 millisecond of time before the cancel takes place and the grab frame routine returns as if has timed out.

### **Usage:**

```

OSStatus err;
//cancel any pending frame grabs..
err = ASCDCAMCancelGrabFrame(theCameraPtr);

```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error

codes.

If a call to "ASDCAMSaveFrameToDisk()", "ASDCAMGrabARGB32Frame()" or "ASDCAMGrabRawFrame()" is in progress when you call "ASDCAMCancelGrabFrame()", then will return a "-9074" as if they had timed out.

---

```
OSStatus ASCDCAMAssignCallback(    void* pTheCamera,
                                   ASC_CallbackProcPtr callBackPtr,
                                   void* userDataPtr);
```

### **Description:**

Assigns a user defined function, which will be called whenever a frame of video is available. The API will not reenter your routine for that camera and will not call you again until it has returned from your callback. If the frame rate is faster than you can process the callbacks, it skips over calling your functions and drops the frames.

The "callBackPtr" is either "NULL", which indicates you no longer want your routine to be called or an "ASC\_CallbackProcPtr" pointer to a function that you want called and is defined as:

```
void myDisplayCallback(ASC_CallbackStruct theCallBackData)
```

The name of your function does not matter, it simply has to accept a "ASC\_CallbackStruct" as parameter and does not return anything.

The "userDataPtr" is a pointer to some information that you want passed to you each time your callback functions is called. This can be a pointer to a WindowRef, pointer to the Camera, pointer to a structure or anything you need.

The "ASC\_CallbackStruct" is defined in the "ASC\_DCAM\_API.h" and returns information on the image (height, width and number of bytes), a pointer to a image buffer, the "userDataPtr" that you passed in to the "ASDCAMAssignCallback()" and TimeStamp (elapsed time since video stream was started and times since last frame arrived) information.

### **Usage:**

```
typedef struct
{
    void *          theCameraPtr;//camera object
    short           theVRefNum;//for fsspec
    long            theParID;//for fsspec
    Boolean          grabThisFrame;//flag to grab frames..
    UInt32           frameNumber;//frame number
    OSStatus         theErr;//keep track of errors
} yourProcessFrameCallBack, *yourProcessFrameCallBackPtr;

void yourGrabFrameCallBack(ASC_CallbackStruct theCallBackData)
{
    OSStatus          err;
    yourProcessFrameCallBackPtr    yourPFCBPtr;
```



```

FSSpec                                theSpec;
Str255                                theFileNamePascal;
char                                  theFileNameC[256];

//extract our ourProcessFrameCallBack pointer..
yourPFCBPtr = (yourProcessFrameCallBackPtr)theCallBackData.userDataPtr;
if (yourPFCBPtr->grabThisFrame == true)//grab the frame to disk
{
    sprintf(theFileNameC,"frame%d.TIFF",ourCFCBPtr->frameNumber);
    //convert C to Pascal..
    CopyCStringToPascal(theFileNameC,theFileNamePascal);
    //create the fsspec..
    err = FSMakeFSSpec(yourPFCBPtr->theVRefNum,
                        yourPFCBPtr->theParID,
                        theFileNamePascal,
                        &theSpec);

    if (err == fnfErr)
    //if it's an fnfErr, no file exists at our location
    //so we are not overwriting another file
    {
        //save the frame to disk..
        err = ASCDCAMSaveFrameToDisk(    theCameraPtr,
                                          theSpec,
                                          kQTFileTypeTIFF,//tiff
                                          kASC_ARGB,//ARGB only..
                                          (UInt32)5000);//wait up to 5 seconds..
    }
    else//something else was returned..
    {
        if (err == 0)//we are trying to overwrite an existing file
        {
            err = 0xFFFFFFFF;// make it something weird as a flag..
        }
    }
    if (err != 0)//failed so stop grabbing frames..
        yourPFCBPtr->grabThisFrame = false;
    yourPFCBPtr->theErr = err;
    yourPFCBPtr->frameNumber++;
}
}

OSStatus                                err;
yourProcessFrameCallBackPtr            yourPFCBPtr;
FSSpec                                theUsersFSSpec;

//your function to get the location you/ the user wish to save frames at..
err = yourGetFSPECLocationToSaveFilesFromUser(&theUsersFSSpec);
if (err != 0)
    return 0;

yourPFCBPtr = (yourProcessFrameCallBackPtr) NewPtrClear(sizeof(yourProcessFrameCallBack));
err = MemError();
if (err == 0)
{
    //initialize the "yourProcessFrameCallBack" structure..
    yourPFCBPtr->theCameraPtr = theCameraPtr;
    yourPFCBPtr->theVRefNum = theUsersFSSpec.vRefNum;
}

```

```

yourPFCBPtr->theParID = theUsersFSSpec.parID;
yourPFCBPtr->grabThisFrame = false;
yourPFCBPtr->frameNumber = 0;
yourPFCBPtr->theErr = 0;
err = ASCDCAMAssignCallback(theCallbackDataPtr->theCameraPtr,
                           (ASC_CallbackProcPtr) yourGrabFrameCallBack,
                           (void*) yourPFCBPtr);

if (err == 0) //Set camera to 640x480 RGB24 video at up to 7.5 fps
    err = ASCDCAMSetFMR(theCameraPtr,0,4,(double)7.5);
    if (err == 0)
{
    //start the isoch stream
    err = ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,true);
    if (err == 0)
    {
        //Start the camera transmitting video..
        err = ASCDCAMStartStopVideo(theCameraPtr,true);
        if (err == 0) //camera is running..
        {
            //Start it grabbing frames..
            yourPFCBPtr->grabThisFrame = true;
            //run for 5 seconds saving frames to disk..
            CFRunLoopRunInMode(kCFRunLoopDefaultMode,5,false);
            ASCDCAMStartStopVideo(theCameraPtr,false);//stop the camera
        }
        //release the IsochResources..
        ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,false);
    }
    ASCDCAMAssignCallback(theCallbackDataPtr->theCameraPtr,NULL,NULL);
}

```

### **Notes and Conditions:**

You need to obtain a reference to the camera using "ASCDCAMOpenACamera()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

Do not alter, modify or dispose of the "ImageBufferPtr" passed to you in the ""ASC\_CallbackStruct" of your Callback Function. Those buffers are internally managed by the API.

Unless the camera is transmitting frames of video, your callback function will never be called.

You should be very careful with the function that you assigning to your callback and should always pass it a "NULL" for the "callBackPtr" as soon as you are finished using it. Your callback function remains assigned even if you deallocate and then reallocate the isochronous channel.

Do not reuse the same callback function with multiple cameras at the same time, especially on multi processor Macs. Even if you keep state flags indicating whether your callback is being reentered, you will still drop frames of video for the other cameras that call your callback. For optimum performance and code safety, allocate a callback for each camera you are operating.

---

```
void * ASCDCAMCreateMovie( void* pTheCamera,
                           double expectedFrameDurationMS,
                           TimeScale trackTimeScale,
                           UInt32 width,
                           UInt32 height,
                           OSType cCCCType,
                           UInt32 bitsPerPixel,
                           UInt32 rowByteWidth,
                           FSSpec fileLocation,
                           OSStatus *theErr);
```

### **Description:**

This API call is used to create QuickTime movie for recording too. It allocates file storage, memory buffers and pthreads for asynchronously saving frames to disk. The creation process is not thread or preemptive safe and should only be called from the main thread of your application.

The "expectedFrameDurationMS" parameter is expected frame duration in milliseconds. This handles a special case for the first frame that is added to the movie. QuickTime movies keep only the duration of each frame and calculating a frames duration requires two frames.

If the "expectedFrameDurationMS" is less than or equal to 0, then first frame that arrives is NOT added to the movie, but the arrival time is kept. When the next frame arrives it is added to the movie and the duration of that frame is calculated using the previous frames time. This will skip the first frame.

If the "expectedFrameDurationMS" is >0, then first frame that arrives is added to the movie and the duration of it is set to the "expectedFrameDurationMS". When the next frame arrives it is added to the movie and the duration of that frame is calculated using the previous frames time. This will keep the first frame.

The reason for this special case is for users that might be doing long exposures and losing the first frame out of the camera is not acceptable, so this allows you to compensate for it.

The "trackTimeScale" parameter specifies the time granularity units for the video track's time scale, which is the number of time samples per second. Generally speaking just use a value of 3000 unless your frame rate is faster than 100fps. At higher frame rates you will want to increase the granularity value and can use the following equation

$$\text{trackTimeScale} = \text{frames per second} * 100; // \text{result must be } > 1 \text{ and } < 0xFFFFFFFF$$

to calculate it. Do not make this "trackTimeScale" too large, as the maximum duration a movie can have is

$$\text{MaxDuration} = 0xFFFFFFFF / \text{trackTimeScale}$$

So if your "trackTimeScale" value is enormous, then the total movie time will be shortened proportionally.

The "width" and "height" parameters are the width and height in pixels of the movie frames you will add.

The "rowByteWidth" parameter allows you to specify the number of bytes in a line of video in the "destPtr" destination buffer, which accounts for any padding at the end of each line in the buffers. This allow the case where you may have converted a frame of RAW video into an ARGB32 PixMapHandle from GWorldPtrs. For Altivec routines, you'll want to ensure the end of each line is padded to a multiple of 16 bytes. If your using a simple buffer as a destination, then you can optionally use a value of 0.

The "fileLocation" is an FSSpec structure indicating where the file will be saved to. You can programatically create your own FSSpec's or used Navigation Services to get one base don the users choice. Existing files will

not be over written.

The "cCCCType" and "bitsPerPixel" parameters indicate the format of the video frames you are adding to the movie. You should get these parameters from the camera using the "ASCDCAMGetFMRLTWHCB()" API call. Additionally if the camera is delivering the Bayer format video data and you wish to record it in that format, then you will want to use the appropriate RGGB, GBRG, BGGR or GRBG format as listed below.

cCCCType	Bit Depth	Video Format
kRawCodecType	8	Monochrome8
kRawCodecType	16	Monochrome16
kRawCodecType	24	RGB24
kRawCodecType	32	ARGB32
kRawCodecType	48	RGB48
k444YpCbCr8CodecType	24	YUV444 * 1
k422YpCbCr8CodecType	16	YUV422
kYUV411PixelFormat	12	YUV411 * 2
kLQ_ASCBAYER_RGGB	8, 12* 3, 16	Low Quality RGGB* 4
kMQ_ASCBAYER_RGGB	8, 12* 3, 16	Medium Quality RGGB* 4
kHQ_ASCBAYER_RGGB	8, 12* 3, 16	High Quality RGGB* 4
kLQ_ASCBAYER_GBRG	8, 12* 3, 16	Low Quality GBRG* 4
kMQ_ASCBAYER_GBRG	8, 12* 3, 16	Medium Quality GBRG* 4
kHQ_ASCBAYER_GBRG	8, 12* 3, 16	High Quality GBRG* 4
kLQ_ASCBAYER_GRBG	8, 12* 3, 16	Low Quality GRBG* 4
kMQ_ASCBAYER_GRBG	8, 12* 3, 16	Medium Quality GRBG* 4
kHQ_ASCBAYER_GRBG	8, 12* 3, 16	High Quality GRBG* 4
kLQ_ASCBAYER_BGGR	8, 12* 3, 16	Low Quality BGGR* 4
kMQ_ASCBAYER_BGGR	8, 12* 3, 16	Medium Quality BGGR* 4
kHQ_ASCBAYER_BGGR	8, 12* 3, 16	High Quality BGGR* 4

\* 1 There is no YUV444 codec in QuickTime for displaying YUV444 video, so it is automatically converted to ARGB32 format and saved in the movie.

\* 2 There is no YUV411 codec in QuickTime for displaying YUV411 video, so it is automatically converted to YUV422 (k422YpCbCr8CodecType) format and saved in the movie.

\* 3 The 12 bit Bayer video frames are automatically converted to Bayer 16 format and saved in the movie.

\* 4 To play the Bayer video movie requires our "ASC\_BAYER.component" which is currently a separate licensable item.

## **Usage:**

See the "ASCDCAMSaveFrameToDisk()" usage example below for details.

### **Notes and Conditions:**

This API call is only available in version 2.0 of the IIDC/DCAM Framework.

You need to obtain a reference to the camera using "ASDCAMOpenACamera()" before using this routine.

If successful it returns a "pMovieData" object. If it fails, it return NULL and a non zero error value will be returned in the "theErr" parameter. See "Appendix A" for a list of common error codes.

---

```
OSStatus ASCDCAMAddMovieFrame(    void* pMovieData,
                                void * framePtr,
                                double timeInMilliseconds);
```

### **Description:**

This can be called from your callback function or any where else and is thread safe.

pMovieData is a movie recording object that was created by ASCDCAMCreateMovie() function call.

The "framePtr" is a pointer to a frame of video to add to this movie.

timeInMilliseconds is the elapsed time in milliseconds that the frame was captured at and must be greater then the previous time or you will receive an -2015 error. Time does not flow backwards.

### **Usage:**

```
FSSpec    fileLocation
OSType    fileFormat;
OSType    displayFormat;
OSStatus  err;
UInt32    format;
UInt32    mode;
double    frameRateDbl;
UInt32    left;
UInt32    top;
UInt32    width;
UInt32    height;
OSType    cCCCType;
UInt32    bitsPerPixel;
void*     theBufferPtr = NULL;
void*     pMovieRecordingObject = NULL;
double    elapsedTimeDouble;
double    frameDurationTimeDouble;
double    recordingTime;

//get the current camera settings..
err = ASCDCAMGetFMRLTWHCB(    theCameraPtr,
                             &format,
```

```

        &mode,
        &frameRateDbl,
        &left,
        &top,
        &width,
        &height,
        &cCCCType,
        &bitsPerPixel);

if (err == 0)
{
    if (format == 7)//format 7 cameras do not have fixed frame rates..
        frameDurationTimeDouble = 1000.00;//fake it out 1 fps..
    else //formats 1 2 and 3 have fixed frame rates
        frameDurationTimeDouble = (1000.00 / frameRateDbl);//time in milliseocnds..
    recordingTime = frameDurationTimeDouble*100.00;
    elapsedTimeDouble = 0.00;
    pMovieRecordingObject = ASCDCAMCreateMovie( theCameraPtr,
                                                frameDurationTimeDouble,
                                                3000,//track time scale
                                                width,
                                                height,
                                                cCCCType,//video format
                                                bitsPerPixel,
                                                (UInt32) ((width*bitsPerPixel)/8),
                                                fileLocation//fsspec
                                                &err);
}
if (err == 0)
{
    theBufferPtr = malloc((width*height*bitsPerPixel)/8);
    if (theBufferPtr == NULL)
        err = -108;//a memFullErr so we are out of ram..
}

while ((err == 0) && (elapsedTimeDouble < recordingTime))
{
    err = ASCDCAMGrabRawFrame( theCameraPtr,
                              theBufferPtr,
                              (UInt32)( frameDurationTimeDouble*2));
    if (err==0)//convert raw frame data into an appropriate RGB GWorld..
        err = ASCDCAMAddMovieFrame( pMovieRecordingObject,
                                    theBufferPtr,
                                    elapsedTimeDouble);

    //increent the elapsed time..
    elapsedTimeDouble = elapsedTimeDouble + frameDurationTimeDouble;
}
if (pMovieRecordingObject != NULL)
    err = ASCDCAMCloseMovie(pMovieRecordingObject);
if(theBufferPtr != NULL)
    free(theBufferPtr);

```

### **Notes and Conditions:**

This API call is only available in version 2.0 of the IIDC/DCAM Framework.

You need to obtain a reference to the pMovieData object using "ASCDCAMCreateMovie()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

If this function is already in use, it will return a kBusyErr (-9074) error.

If you run out of disk space while recording, you will receive a dskFulErr (-34) error and you can not record any more video to the movie. The movie is still valid after receiving this error.

---

OSStatus **ASCD CAMCloseMovie**(void\* pMovieData);

**Description:**

Call this to stop the movie recording, close off the files and to dispose of the pMovieData object that was allocated by "ASCD CAMCreateMovie()" NEVER dispose of the pMovieData object yourself as that will result in memory leaks and will leave temporary files open. ALWAYS pass the pMovieData object to ASCDCAMCloseMovie() so that it can clean up and dispose of everything for you. This routine is not thread safe and should always be called from the main thread of your application.

**Usage:**

See the ASCDCAMSaveFrameToDisk() usage example above for details.

**Notes and Conditions:**

This API call is only available in version 2.0 of the IIDC/DCAM Framework.

You need to obtain a reference to the pMovieData object using "ASCD CAMCreateMovie()" before using this routine.

The returned value will be 0 if successful, negative otherwise. See "Appendix A" for a list of common error codes.

---

# Simple Code Samples

## Sample 1

This Sample shows:

- how to locate a camera
- open it for exclusive use
- how to discover what the camera supports for Fixed Formats and Modes,
- how to start the camera transmitting video
- how to grab a tiff frame to disk
- how to clean up everything

```
/*
This routine finds the first available Fixed Size Format and Mode for
this DCAM camera and then selects it auto magically.
It ignores the Format 7 Variable Size formats and modes..
*/
```

```
OSStatus useFirstAvailableFormatMode(void * theCameraPtr)
{
    OSStatus          err;
    short             theMode;
    CameraCapabilitiesStruct theProperties;
    Boolean           done;

    err = ASCDCAMGetCameraProperties(theCameraPtr, &theProperties);
    if (err!=0)
        return err;

    done = false;
    theMode = 0;
    if (theProperties.format0 == true)//supports format 0
    {
        //pick a Mode
        while ((done == false) && (theMode<8))
        {
            if (theProperties.format0Modes[theMode] ==true)
                done = true;
            else
                theMode++;
        }
        if (done==true)//60.00 means as fast as possible
            err = ASCDCAMSetFMR(theCameraPtr,0,theMode,(double)60.00);
        else
            err = 0xFFFFFFFF;
        return err;
    }

    if (theProperties.format1 == true)//supports format 1
    {
        //pick a Mode
        while ((done == false) && (theMode<8))
        {
            if (theProperties.format1Modes[theMode] ==true)
                done = true;
            else
```



```

        theMode++;
    }
    if (done==true)//60.00 means as fast as possible
        err = ASCDCAMSetFMR(theCameraPtr,1,theMode,(double)60.00);
    else
        err = 0xFFFFFFFF;
    return err;
}
if (theProperties.format2 == true)//supports format 2
{
    //pick a Mode
    while ((done == false) && (theMode<8))
    {
        if (theProperties.format2Modes[theMode] ==true)
            done = true;
        else
            theMode++;
    }
    if (done==true)//as fast as possible
        err = ASCDCAMSetFMR(theCameraPtr,2,theMode,(double)60.00);
    else
        err = 0xFFFFFFFF;
    return err;
}
//Only format 7 is supported and we don't want to do that one for this example..
return (OSStatus) 0xFFFFFFFF;
}

static          OSStatus          myGetDefaultLocation(short *theVRefNum, long
*theParID)
{
//this locates the Folder that the Application was launched from
//and returns the location that it's (vrefnum and parId) are from.
    OSStatus theErr;
    ProcessSerialNumber thePSN;
    ProcessInfoRec theInfo;
    FSSpec theSpec;

    thePSN.highLongOfPSN = 0;
    thePSN.lowLongOfPSN = kCurrentProcess;

    theInfo.processInfoLength = sizeof(theInfo);
    theInfo.processName = NULL;
    theInfo.processAppSpec = &theSpec;

    /* Find the application FSSpec */
    theErr = GetProcessInformation(&thePSN, &theInfo);

    //recursively move out through the directories of the applicaiton bundle..
    // Find the "Contents" (the parent of the "MacOS" directory)
    if (theErr == noErr)
        theErr = FSMakeFSSpec(theSpec.vRefNum, theSpec.parID, "\p", &theSpec);

    // Find the parent Bundle folder (the parent of the "Contents" directory)
    if (theErr == noErr)
        theErr = FSMakeFSSpec(theSpec.vRefNum, theSpec.parID, "\p", &theSpec);

    // Find the parent of the Bundle folder

```

```

if (theErr == noErr)
    theErr = FSMakeFSSpec(theSpec.vRefNum, theSpec.parID, "\p", &theSpec);

if (theErr == noErr)
{
    /* Return the folder which contains the application package */
    *theVRefNum = theSpec.vRefNum;
    *theParID = theSpec.parID;
}
return theErr;
}

int main (void)
{
    OSStatus          err;
    dcamCameraList theCameraList;
    void *             theCameraPtr;
    FSSpec             theSpec;
    short              theVRefNum;
    long               theParID;
    //get list of cameras attached..

    err = myGetDefaultLocation(&theVRefNum,&theParID);
    if (err!= 0)
        return 0;//bail..
    err = FSMakeFSSpec(theVRefNum, theParID, "\pATiffFile.tif", &theSpec);
    if (err!= -43)//a -43 indicates the file does not exist at this location,
        return 0;//if it's anything else, Quit..
    err = ASCDCAMGetListOfCameras(&theCameraList);
    if (err!= 0)//an error occured..
        return 0;//bail..
    if (theCameraList.numCamerasFound == 0)//no cameras.
        return 0;//bail..
    //open the first camera found - we assume it is not in use..

    theCameraPtr = ASCDCAMOpenACamera(theCameraList.theCameras[0].uniqueID);
    if (theCameraPtr!=NULL)//we have it exclusively now..
    {
        err = useFirstAvailableFormatMode(theCameraPtr);
        if (err == 0)
        {
            //start the isoch stream
            err = ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,true);
            if (err == 0)
            {
                //Start the camera transmitting video..
                err = ASCDCAMStartStopVideo(theCameraPtr,true);
                if (err == 0)
                {
                    //camera is running..
                    //run the camera for 2 seconds
                    //to allow any auto controls to adjust themselves
                    CFRRunLoopRunInMode( kCFRunLoopDefaultMode,2, false );
                    //Grab a frame..
                    err = ASCDCAMSaveFrameToDisk(theCameraPtr,
                                                theSpec,
                                                kQTFileTypeTIFF,//tiff
                                                kASC_ARGB,//ARGB only..
                                                (UInt32)1000);//wait up to 1 second..

                    ASCDCAMStartStopVideo(theCameraPtr,false);//stop the camera
                }
            }
        }
    }
}

```

```

    }
    //release the Isoch Resources..
    ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,false);
}
}
//all done so release the camera and System Resources
ASCDCAMCloseACamera(theCameraPtr);
}
return 0;
}

```

## Sample 2

Sample 2 builds on what was shown in the previous sample and covers:

- how to create a callback function for displaying video in a window
- how to grab a tiff frame to disk
- how to clean up everything

//our callback structure..

```

typedef struct
{
    Boolean          shouldDisplay;
    Boolean          isBayerData;
    OSStatus         theErr;//keep track of errors
    UInt32           theWidth;
    UInt32           theHeight;
    UInt32           bitsPerPixel;
    OSType           theDisplayFormat;
    OSType           fourCCType;
    Rect             GWorldBounds;
    void *           theCameraPtr;//camera object
    WindowRef        thePreviewWindow;
    GWorldPtr        theGWorld;
} yourProcessFrameCallBack, *yourProcessFrameCallBackPtr;

```

```

static void yourDisplayCallback(ASC_CallbackStruct theCallBackData)

```

```

{
    CGrafPtr         theWindowPort;
    OSStatus         err = noErr;
    yourProcessFrameCallBackPtr theCallbackDataPtr;
    PixMapHandle     hPixMap;
    Rect             theWinRect;
    Ptr              baseAddress;
    UInt32           pixRowBytes;

    if ((theCallBackData.userDataPtr != NULL) && (theCallBackData.ImageBufferPtr != NULL))
    {
        theCallbackDataPtr = (yourProcessFrameCallBackPtr) theCallBackData.userDataPtr;
        if (theCallbackDataPtr->shouldDisplay == true)
        {
            hPixMap = GetGWorldPixMap(theCallbackDataPtr->theGWorld);
            if (LockPixels(hPixMap) == true)
            {
                baseAddress = GetPixBaseAddr(hPixMap);
                pixRowBytes = (UInt32) GetPixRowBytes(hPixMap);
            }
        }
    }
}

```

```

if (theCallbackDataPtr->isBayerData == true)
{
    //we need to treat this as a Bayer frame..
    err = ASCDCAMConvertRawFrameToARGB32(theCallbackDataPtr->theCameraPtr,
        theCallBackData.ImageBufferPtr,
        theCallbackDataPtr->theWidth,
        theCallbackDataPtr->theHeight,
        theCallbackDataPtr->theDisplayFormat,
        theCallbackDataPtr->bitsPerPixel,
        baseAddress,
        pixRowBytes);
}
else//treat this as a "normal" image..
{
    err = ASCDCAMConvertRawFrameToARGB32(theCallbackDataPtr->theCameraPtr,
        theCallBackData.ImageBufferPtr,
        theCallbackDataPtr->theWidth,
        theCallbackDataPtr->theHeight,
        theCallbackDataPtr->fourCCType,
        theCallbackDataPtr->bitsPerPixel,
        baseAddress,
        pixRowBytes);
}
if (err == 0)
{
    theWindowPort = GetWindowPort(theCallbackDataPtr->thePreviewWindow);
    GetWindowPortBounds(theCallbackDataPtr->thePreviewWindow, &theWinRect);
    //in case the window has a differnt size now.
    SetPortWindowPort(theCallbackDataPtr->thePreviewWindow);
    if (LockPortBits(theWindowPort) == noErr)
    {
        CopyBits(((BitMapPtr)*hPixMap),
            GetPortBitMapForCopyBits(theWindowPort),
            &theCallbackDataPtr->GWorldBounds,
            &theWinRect,
            srcCopy,
            NULL);
        UnlockPixels(hPixMap);
        if (QDIsPortBuffered(theWindowPort))//this port is buffered..
            QDFlushPortBuffer(theWindowPort,NULL);//update the window..
        UnlockPortBits(theWindowPort);
    }
    else//failed LockPortBits
    {
        UnlockPixels(hPixMap);
        err = 0xFFFFFFFF;
    }
}
else//failed..
    UnlockPixels(hPixMap);
}
else//failed LockPixels
    err = 0xFFFFFFFF;
theCallbackDataPtr->theErr = err;
if (theCallbackDataPtr->shouldDisplay == true)
    theCallbackDataPtr->shouldDisplay = (err == 0);
}
}
}

```

```

OSStatus SetUpDisplayCallback(void *theCameraPtr, yourProcessFrameCallBackPtr yPFCBPtr)
{
    Rect        windowRect;
    Rect        deviceRect;

    OSStatus err;
    UInt32      format;
    UInt32      mode;
    double      frameRateDbl;
    UInt32      left;
    UInt32      top;

    //inititalize everything..
    yPFCBPtr->theCameraPtr = theCameraPtr;
    yPFCBPtr->thePreviewWindow = NULL;
    yPFCBPtr->theGWorld = NULL;
    yPFCBPtr->shouldDisplay = false;
    yPFCBPtr->theErr = 0;
    err = ASCDCAMGetFMRLTWHCB(yPFCBPtr->theCameraPtr,
                              &format,
                              &mode,
                              &frameRateDbl,
                              &left,
                              &top,
                              &yPFCBPtr->theWidth,
                              &yPFCBPtr->theHeight,
                              &yPFCBPtr->fourCCType,
                              &yPFCBPtr->bitsPerPixel);

    if (err != 0)
        return err;//bail
    //if your FW camera does bayer video for this format and mode,set
    // isBayerData to true
    yPFCBPtr->isBayerData = false;
    //Change this to whatever kLQ_ASCBAYER_RGGB or RGGB pattern the FW Camera uses
    //for the Bayer data..
    if (yPFCBPtr->isBayerData == true)
        yPFCBPtr->theDisplayFormat = kLQ_ASCBAYER_RGGB;
    else
        yPFCBPtr->theDisplayFormat = kASC_ARGB;
    // find the best monitor for the window
    GetBestDeviceRect(NULL, &deviceRect);
    SetRect(&windowRect,0,0,
            (short) (yPFCBPtr->theWidth),
            (short) (yPFCBPtr->theHeight));

    SetRect(&yPFCBPtr->GWorldBounds,0,0,
            (short)(yPFCBPtr->theWidth),
            (short) (yPFCBPtr->theHeight));

    //create the GWorld Buffer for the ARGB32 frames
    err = NewGWorld(&(yPFCBPtr->theGWorld),32,&yPFCBPtr->GWorldBounds,NULL,NULL,0);
    if (err == noErr)
    {
        // put the window near the top left of this monitor
        OffsetRect(&windowRect,
                  (short)(deviceRect.left + 12),
                  (short)(deviceRect.top + 48));
    }
}

```



```

        RunApplicationEventLoop();//loop forever until the user Command-Quits..
        yPFCBPtr->shouldDisplay = false;
        //stop the callback..
        ASCDCAMAssignCallback(theCameraPtr, NULL,NULL);
    }
    //stop the camera
    ASCDCAMStartStopVideo(theCameraPtr,false);
}
//release the Isoch System Resources and buffers..
ASCDCAMAllocateReleaseIsochChannel(theCameraPtr,false);
}
}
//all done so release the camera and System Resources
ASCDCAMCloseACamera(theCameraPtr);
if (yPFCBPtr != NULL)//clean up the callback data..
{
    if (yPFCBPtr->thePreviewWindow != NULL)
        DisposeWindow(yPFCBPtr->thePreviewWindow);
    if (yPFCBPtr->theGWorld != NULL)
        DisposeGWorld(yPFCBPtr->theGWorld);
    DisposePtr((Ptr) yPFCBPtr);
}
}
return 0;
}

```

### **Sample 3**

Sample 3 builds on what was shown in the previous samples and covers:

- how to create a callback function for displaying video in a window
- how to preview video in a window and record it to disk in a movie at the same time.
- how to clean up everything

See the "ASC\_DCAM Sample3" project in the "ASC\_DCAM Simple Examples" folder.

## "APPENDIX A" - Error Codes

Rather than reinvent the wheel, we have used Apple's existing error codes (the ones in "()" brackets ) where ever possible. The common errors that you will most likely see are listed below.

Error#	Description
-50	One or more of the parameters passed to an API function are out of range ( <i>paramErr</i> ). This usually means you have tried to select a format or mode that the camera does not support.
-108	Not enough memory to complete the operation ( <i>memFullErr</i> ).
-128	An API function has exceeded your specified time out value ( <i>userCanceledErr</i> ) while waiting for something to happen.
-2201	Requested feature is not supported by the hardware ( <i>digiUnimpErr</i> ). This usually means you have tried to select a format - mode that the camera does not support or have tried using a Control that the camera does not physically support.
-9074	Your Callback function took longer than 10 seconds to return ( <i>kBusyErr</i> ). If you pause ( <i>ASDCAMResumePauseVideo</i> ) or stop ( <i>ASDCAMStartStopVideo</i> ) the video flow and your assigned callback function is still being used after 10 seconds has elapsed, you will receive this error. Your callback must first return before the API will allow you to stop the stream.
-536870208	No camera is connected ( <i>kIOReturnNoDevice</i> or 0xe00002c0), which means you probably passed a NULL camera to an API routine or the camera may have been physically disconnected.

Table 1 Common Error Codes.

Other errors for FireWire, Kernel, QuickTime, VMAllocate and other OSX technologies will most likely be listed in the Apple "MacErrors.h", "IOReturn.h" and "IOFireWireFamilyCommon.h" headers.

## "APPENDIX B" - Fixed Formats, Modes and Frame Rate

Mode	Format 0 Image Size and Format	Format 1 Image Size and Format	Format 2 Image Size and Format
0	160x120 YUV444	800x600 YUV422	1280x960 YUV422
1	320x240 YUV422	800x600 RGB24	1280x960 RGB24
2	640x480 YUV411	800x600 Monochrome8	1280x960 Monochrome8
3	640x480 YUV422	1024x768 YUV422	1600x1200 YUV422
4	640x480 RGB24	1024x768 RGB24	1600x1200 RGB24
5	640x480 Monochrome8	1024x768 Monochrome8	1600x1200 Monochrome8
6	640x480 Monochrome16	800x600 Monochrome16	1280x960 Monochrome16
7	Undefined	1024x768 Monochrome16	1600x1200 Monochrome16

Table 2 "Fixed" Formats and Modes to Image Size and Pixel Sizes.



FrameRate	Frames Per Second
0	1.875
1	3.750
2	7.500
3	15.00
4	30.00
5	60.00
6	120.00
7	240.00

**Table 3** "Fixed FrameRates" for Formats 0,1 and 2