HAMAMATSU

PHOTONIC SYSTEMS

A DIVISION OF HAMAMATSU CORPORATION

DCAMAPI Library for LabVIEW

June 2004

Version 2.3

HAMAMATSU

Homepage Address http://www.hamamatsu.com

HAMAMATSU PHOTONICS K.K., Systems Division
812 Joko-cho, Hamamatsu City, 431-3196, Japan, Telephone: (81)53-431-0124, Fax: (81)53-435-1574, E-mail:export@sys.hpk.co.jp

U.S.A. and Canada: Hamamatsu Photonic Systems: 360 Foothill Road, Bridgewater, N.J. 08807-0910, U.S.A., Telephone: (1)908-231-1116, Fax: (1)908-231-0852, E-mail: usa@hamamatsu.com
Germany: Hamamatsu Photonics Deutschland GmbH: Arzbergerstr. 10, D-82211 Herrsching am Ammersee, Germany, Telephone: (49)8152-375-0, Fax: (49)8152-2658, E-mail: info@hamamatsu.de
France: Hamamatsu Photonics France S.A.R.L.: 8, Rue du Saule Trapu, Parc du Moulin de Massy, 91882 Massy Cedex, France, Telephone: (33)1 69 53 71 00, Fax: (33)1 69 53 71 10, E-mail: infos@hamamatsu.fr
United Kingdom: Hamamatsu Photonics UK Limited: 2 Howard Court, 10 Tewin Road Welwyn Garden City Hertfordshire AL7 1BW U.K., Telephone: (44)0 1707-294888, Fax: (44)0 1701-325777, E-mail: info@hamamatsu.co.uk
North Europe: Hamamatsu Photonics Norden AB: Smidesvägen 12, SE-171-41 Solna, Sweden, Telephone: (46)8-509-031-00, Fax: (46)8-509-031-01, E-mail: info@hamamatsu.se
Italy: Hamamatsu Photonics Italia S.R.L.: Strada della Mois, 1/E 20020 Arese (Milano), Italy, Telephone: (39)02-935 81 733, Fax: (39)02-935 81 741, E-mail: info@hamamatsu.it

Overview

This DCAMAPI VI library is designed for LabVIEW 7.0 to control a Hamamatsu camera and gather image data. Because it is built on DCAMAPI v2.1.3, any program created with it for any Hamamatsu camera can easily be used for another Hamamatsu camera with little to no changes in the code. And because it is built for LabVIEW, a programmer will be able to create new virtual instruments in a relatively short amount of time.

DCAMAPI is an interface for Hamamatsu cameras. A program that uses the DCAMAPI interface will gain full control of the capabilities of any Hamamatsu camera. It will allow the user to automatically detect the type of camera and the specific capabilities that it offers. If a program using DCAMAPI is designed properly, it will be able to detect all current and future cameras as well as support all of their features. It is recommended that the user understands the DCAMAPI concept before creating applications with this library. For more information on DCAMAPI, please review the DCAMAPI general reference manual with your SDK software.

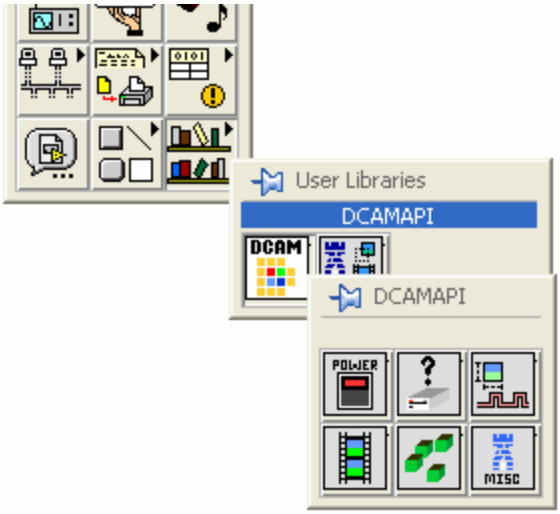
This library consists of many functions to control a Hamamatsu camera. However, this library does not contain any image processing functions. It is strongly recommended to use National Instruments IMAQ Vision 6.1 or higher for processing data.

System Requirement

- Microsoft Windows 2000 or Microsoft Windows XP
- Hamamatsu camera with compatible DCAM module
- National Instruments LabVIEW 7 Express or higher
- National Instruments IMAQ Vision 6.1 or higher (strongly recommended)

Installation

- Be sure that you have LabVIEW 7.0 or higher installed properly on your computer. Also be sure that LabVIEW is not running.
- Insert the DCAM-SDK CD into your CD-ROM drive.
- If the auto-run program does not startup, run SETUP.EXE located on the CD.
- From the setup dialog, select LabVIEW VI Library.
- The following dialogs will allow you to customize your installation.
- After installation is completed, the new VIs will be available in your control box under user libraries.



DCAMAPI Functions

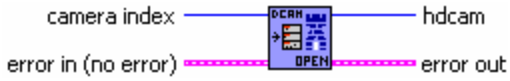
The following are the VIs included with this library. You will notice that there is a VI for every DCAMAPI function. These VIs are wrappers for DCAMAPI and many tasks that can be done using DCAMAPI can be done with this library. The following is a brief description of these functions. For more detailed information, please refer to the DCAMAPI general reference manual found on your DCAM-SDK CD.

Initialize Functions

DCAM_INIT.VI

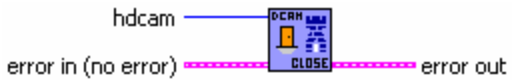
This function will initialize DCAM and it will find all supported cameras that are connected to your computer. This will return the number of cameras that were installed. This function must be called before any cameras can be opened.

DCAM_OPEN.VI



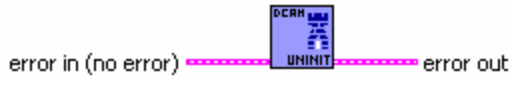
This initializes the specified camera determined by the index input. DCAM_INIT will provide the number of cameras available. If successful, it will return a camera handle HDCAM which will be used by other VIs in this library to access the camera. This will also set the camera to UNSTABLE state.

DCAM_CLOSE.VI



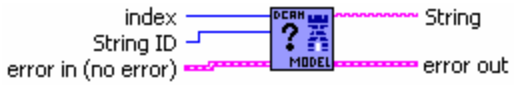
Frees all memory used for the camera and releases the handle. Once this function is called, the camera handle will become invalid and cannot be used by other functions. This function will work if the camera is in BUSY state but it is recommended to call this function in STABLE or UNSTABLE state.

DCAM_UNINIT.VI



This function frees all memory and resources used by DCAM. It is recommended that all opened cameras are closed before executing this function. Once this function is called, no more cameras can be opened for use.

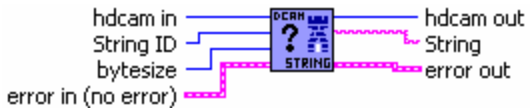
DCAM_GETMODELINFO.VI



This is used to get information on a certain camera such as camera type, serial number, and firmware version. DCAM_INIT must be called before this function. This function does not require a camera handle. To select the camera to check, you must supply the index number. Create a constant or control from the String ID terminal to get a list of valid IDs.

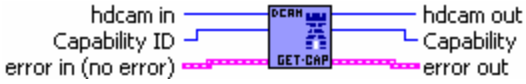
Camera Information Functions

DCAM_GETSTRING.VI



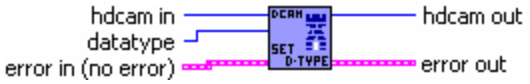
This is used to get information on a certain camera such as camera type, serial number, and firmware version. Unlike DCAM_GETMODELINFO this must be used after DCAM_OPEN because this one requires a camera handle. Create a constant or control from the String ID terminal to get a list of valid IDs.

DCAM_GETCAPABILITY.VI



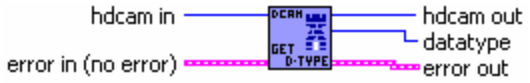
This returns the capabilities of the camera including functions, data types, and bits types. Functions include binning modes, trigger modes, and user memory support. Create a constant or control from the Capability ID terminal to get a list of valid IDs.

DCAM_SETDATATYPE.VI



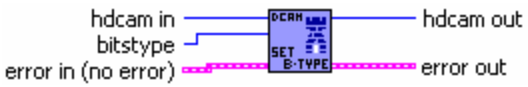
This function allows you to switch data types if the camera supports multiple data types. Create a constant or control from the Datatype terminal to get a list of data type IDs. However, not all of these IDs are valid for each camera. DCAM_GETCAPABILITY will help you in determining what data types are available for your camera.

DCAM_GETDATATYPE.VI



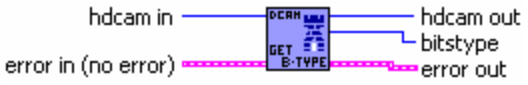
This returns the value of the current data type. The value returned will be numeric and it may be necessary to use the global variables to process this data.

DCAM_SETBITSTYPE.VI



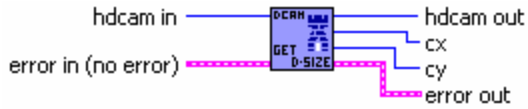
This sets the display type of the image if the camera supports multiple display types. This is for use with the function DCAM_LOCKBITS. Create a constant or control from the Bitstype terminal to get a list of bits type IDs. However, not all of these IDs are valid for each camera. DCAM_GETCAPABILITY will help you in determining what data types are available for your camera.

DCAM_GETBITSTYPE.VI



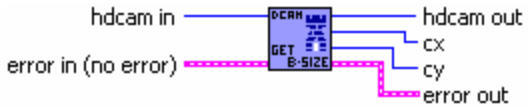
This is similar to DCAM_GETDATATYPE. This returns the value of the current bits type. The value returned will be numeric and it may be necessary to use the global variables to process this data.

DCAM_GETDATASIZE.VI



This function returns the dimensions of the data image. For some cameras, the display size and the data size are different.

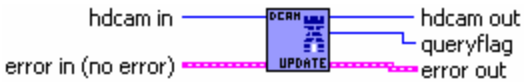
DCAM_GETBITSSIZE.VI



This function returns the dimensions of the display image. For some cameras, the display size and the data size are different.

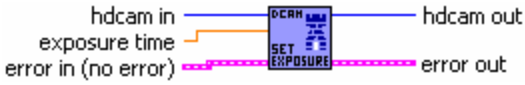
Parameter Functions

DCAM_QUERYUPDATE.VI



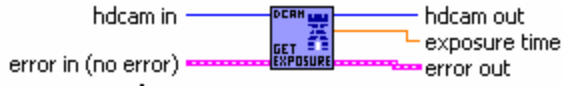
Determines if values have been modified since the last time DCAM_QUERYUPDATE has been called and returns the values with the queryflag output. It is important to adjust your application according to these modifications. For example, if you adjust binning settings, the camera may internally adjust other settings such as exposure time. DCAM_QUERYUPDATE will return DCAM_UPDATE_EXPOSURE to let you know that exposure was changed. The value returned will be numeric and it may be necessary to use the global variables to process this data.

DCAM_SETEXPOSURETIME.VI



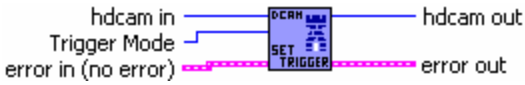
This allows you to modify the exposure time in seconds of the camera. Depending on the camera, the exposure time that you specify may not be the same exposure time actually set. If necessary, check the actual time set using DCAM_GETEXPOSURETIME.

DCAM_GETEXPOSURETIME.VI



This returns the value of the current exposure time in seconds.

DCAM_SETTRIGGERMODE.VI

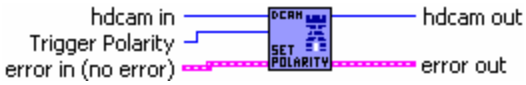


This function allows you to switch the current trigger mode. Create a constant or control from the Mode terminal to get a list of trigger mode IDs. However, although they are listed, it may or may not be a valid ID for your camera.

DCAM_GETTRIGGERMODE.VI

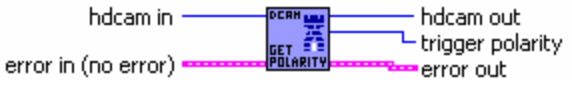
This returns the value of the current trigger mode. The value returned will be numeric and it may be necessary to use the global variables to process this data.

DCAM_SETTRIGGERPOLARITY.VI



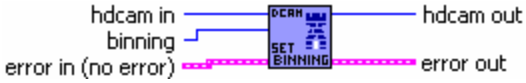
This will select the polarity of the trigger if the camera has been set to external trigger mode. This function will fail if the camera does not support external trigger. Create a constant or control from the Polarity terminal to get a list of valid IDs.

DCAM_GETTRIGGERPOLARITY.VI



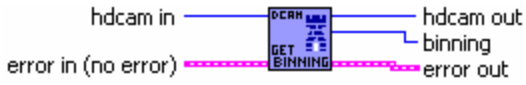
This returns the value of the current polarity. This function will fail if the camera does not support external trigger.

DCAM_SETBINNING.VI



This function will adjust the binning of the camera. Adjusting the binning will also change the output data size, but the pixel depth will remain the same.

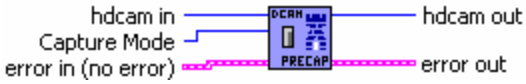
DCAM_GETBINNING.VI



This will return the current binning mode of the camera.

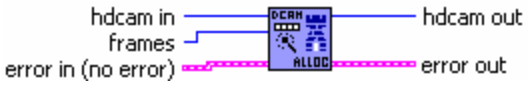
Capturing Functions

DCAM_PRECAPTURE.VI



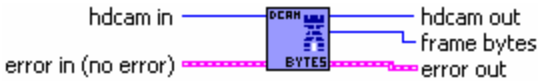
Sets the capture mode and prepares resources for capturing. This also changes the camera status to STABLE state. Create a constant or control from the Capture Mode terminal to get a list of valid IDs.

DCAM_GETDATARANGE.VI



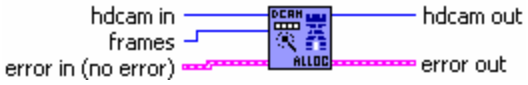
This function will return the minimum and maximum values possible for the data pixels.

DCAM_GETDATAFRAMEBYTES.VI



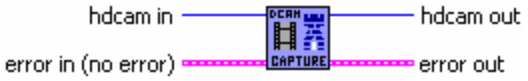
This determines the amount of bytes required for a single frame of data. This is useful when using DCAM_ATTACHBUFFER.

DCAM_ALLOCFRAME.VI



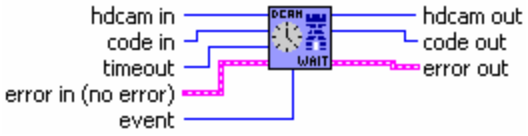
Allocates the proper amount of memory for a data buffer depending on the number of frames requested and changes the camera status from STABLE to READY. Capturing does not start at this time. This function will fail if called while camera is not in STABLE state.

DCAM_CAPTURE.VI



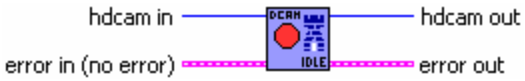
This starts capturing images. Camera status must be READY before using this function. If the camera is in SEQUENCE mode, the camera will capture images repeatedly. If the camera is in SNAP mode, the camera will capture only the number of images for frames that were allocated.

DCAM_WAIT.VI



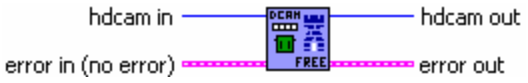
This waits for a user specified event to be generated. Once the event has been signaled, this function will return. Create a constant or control from the Code In terminal to get a list of valid IDs. If the elapsed waiting time exceeds the value set as timeout, then the function will return.

DCAM_IDLE.VI



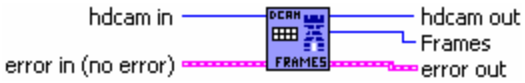
This function will stop the capturing of images. If the camera is in BUSY state, it will set it to READY state.

DCAM_FREEFRAME.VI



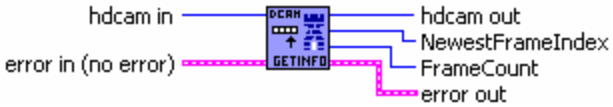
This frees up the memory allocated from DCAM_ALLOCFRAME. Also sets the status of the camera from READY state to STABLE state. This will fail if the camera is in BUSY state when called.

DCAM_GETFRAMECOUNT.VI



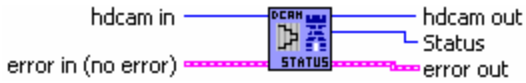
Returns the number of frames allocated in memory by either DCAM_ALLOCFRAME or by DCAM_ATTACHBUFFER. This function will fail is no frames have been allocated.

DCAM_GETTRANSFERINFO.VI



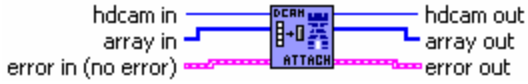
Returns the number of frames captured and the index of the most recent captured frame

DCAM_GETSTATUS.VI



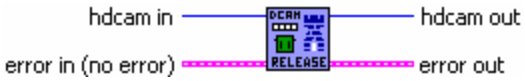
This returns the state of the current camera operation.

DCAM_ATTACHBUFFER.VI



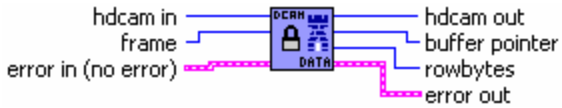
This allows the user to attach his/her own data buffer instead of using DCAM_ALLOCFRAME and using the DCAM buffer. This function accepts an array of pointers to image buffers. If used, this will take the place of DCAM_ALLOCFRAME and set the camera from STABLE to READY state.

DCAM_RELEASEBUFFER.VI



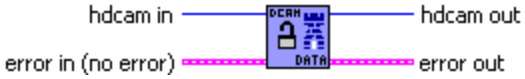
This function works similar to DCAM_FREEFRAME except that this is used when using DCAM_ATTACHBUFFER. This will set the camera from READY to STABLE state.

DCAM_LOCKDATA.VI



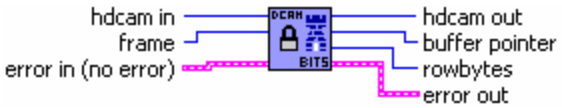
This locks a frame in the data buffer that will be accessed by the user. When a frame is locked, the capture thread will not write to it. Calling this function will also unlock any existing locked frames. If you set the frame input as -1, you will lock the most recently received frame.

DCAM_UNLOCKDATA.VI



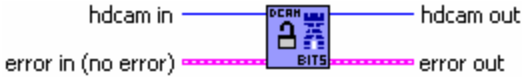
If a framed has been locked by DCAM_LOCKDATA, this function will unlock it.

DCAM_LOCKBITS.VI



This works similar to DCAM_LOCKDATA except that the buffer data has been altered for display. For example, 16 bit images are converted to 8 bit images and flipped for easy use in Windows API functions. If you set the frame input as -1, you will lock the most recently received frame.

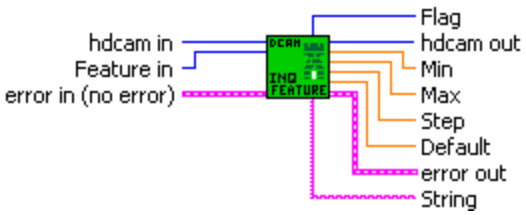
DCAM_UNLOCKBITS.VI



If a framed has been locked by DCAM_LOCKBITS, this function will unlock it.

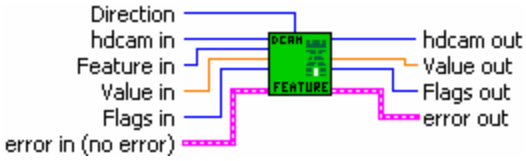
Extended Functions

DCAM_FEATUREINQ.VI



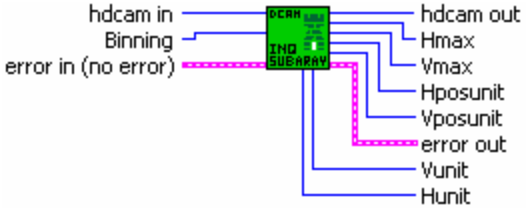
This function gives you to ability to dynamically determine specific details of certain features of the camera. These include minimum value, maximum value, stepping value, and default value. Create a constant or control from the Feature terminal to get a list of IDs. However, not all of these IDs may be valid for your camera.

DCAM_FEATURE.VI



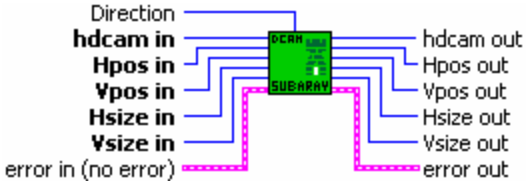
This gives you the ability to set many different features of a camera that you cannot control with the basic functions. With this, you also have the ability to get the current value of the feature. Because the camera will sometimes set a different value than you specify, it is best to use the SETGET option on the Direction terminal so the function will return the value of the feature that you just set. Create a constant or control from the Feature terminal to get a list of IDs. However, not all of these IDs may be valid for your camera.

DCAM_SUBARRAYINQ.VI



This determines the sub-array capabilities of the camera at the specified binning mode. This information is necessary to set valid parameters for DCAM_SUBARRAY.

DCAM_SUBARRAY.VI

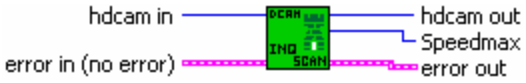


This function gives you the ability to set a new subarray of the capture. Because setting a new subarray will change the image size, this function should not be called in READY or BUSY state. And this will change the camera state to UNSTABLE. This function also allows you to get the current subarray settings.

DCAM_READOUTTIME.VI

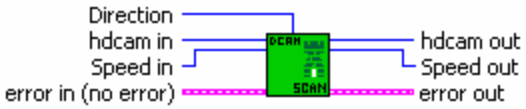
This function will return the current readout time of the camera. This function is not supported by all cameras.

DCAM_SCANMODEINQ.VI



Certain cameras have different scan modes available. Faster speeds allow you to gather more frames while the slower speeds will allow you to get data with less readout noise. Some cameras even have a higher bit depth at the slower speeds. This function will tell you the number of scan speeds available.

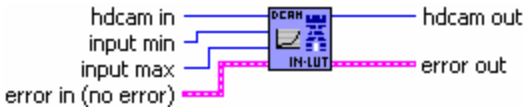
DCAM_SCANMODE.VI



Certain cameras have different scan modes available. Faster speeds allow you to gather more frames while the slower speeds will allow you to get data with less readout noise. Some cameras even have a higher bit depth at the slower speeds. This function will allow you to set a new scan speed and/or determine the current scan mode of the camera.

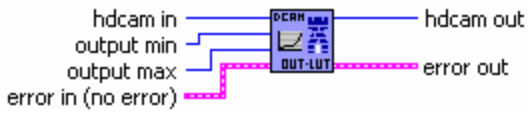
Miscellaneous Functions

DCAM_SETBITSINPUTLUTRANGE.VI



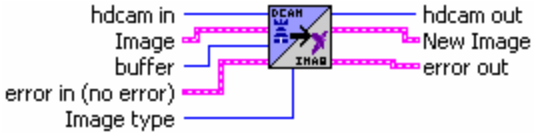
This sets up a lookup table for the display image. This function only affects the image created using DCAM_LOCKBITS.

DCAM_SETBITSOUTPUTLUTRANGE.VI



This sets up a lookup table for the display image. This function only affects the image created using DCAM_LOCKBITS.

DCAM2IMAQ.VI



IMAQ Vision from National Instruments contains many powerful image analysis tools. To take advantages of these tools, this DCAM library comes with some additional functions that can convert DCAM data to IMAQ Vision compatible data. This converts a DCAM image pointer to an IMAQ Vision image cluster that can be used with the IMAQ Vision VIs.

GETIMAQTYPE.VI

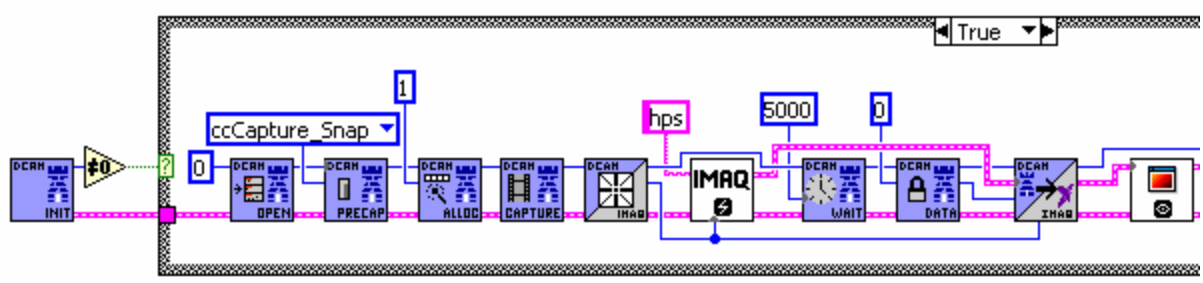
IMAQ Vision from National Instruments contains many powerful image analysis tools. To take advantages of these tools, this DCAM library comes with some additional functions that can convert DCAM data to IMAQ Vision compatible data. This determines what IMAQ Vision image type the camera is capable of. The value returned is useful for the IMAQ Create and DCAM2IMAQ functions.

GLOBALS

Many of the DCAM-API functions accept inputs that are bit masks. Some of the functions can accept several different numeric inputs but those numbers could vary. To assist in this particular situation in C++, programmers would use DEFINES to assign a numeric value to text. Global variables in LabVIEW works similar to DEFINES and make programming simpler and easier to read. **NOTE:** Do not change any of these preset values.

Examples

Single Frame Snap



To snap a single frame, we first need to initial DCAMAPI with DCAM_INIT. If a camera is present in the system, we can then call DCAM_OPEN so we can use this camera. An input of 0 is used to open the first camera found on the system. This will return a handle which will be used by the other VIs.

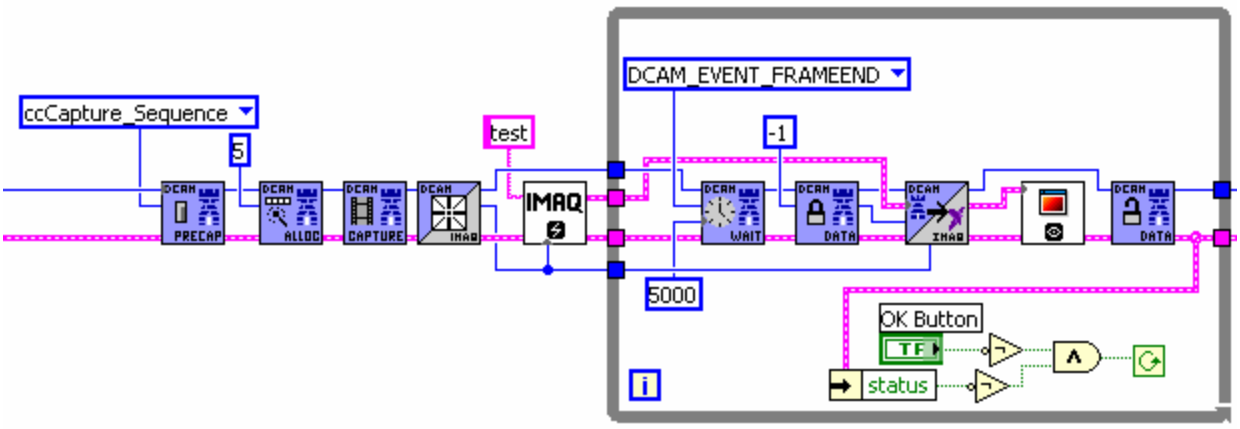
To setup the capture, call DCAM_PRECAPTURE. For a snap, we are using the input ccCapture_Snap. Next we call DCAM_ALLOCFRAME to specify the amount of memory the computer should use for the capture. Because we only want one image, we use 1 for the input. Finally, we can call DCAM_CAPTURE to begin the acquisition.

We must now setup an IMAQ buffer for the new image. GETIMAQTYPE will determine the right type of data that IMAQ requires. We call IMAQ Create to create a new IMAQ image buffer which will be used when processing the frame data.

Before we can use any image data, we must first wait for the image to finish exposing and copying to the DCAMAPI memory. We use DCAM_WAIT to wait for the next available frame. We use DCAM_EVENT_FRAMEEND for the Code In input to state that we want the function to return after the next frame has been received. And we use 5000 for timeout to specify 5 seconds to wait. After the function returns, we call DCAM_LOCKDATA to retrieve the data. An input of 0 is used to get the first frame. This function will return a pointer to the data.

IMAQ Vision tools cannot directly use the DCAMAPI frame pointer. We use DCAM2IMAQ to convert the DCAM data to IMAQ data so it can be used with other IMAQ functions such as IMAQ Display.

Preview Capture



To create a preview capture, we would take need to setup similar to a snap. In this example, we call DCAM_PRECAPTURE. This time we use ccCapture_Sequence. We then call DCAM_ALLOCFRAME with an input of 5. What this does is it will continuously capture images. When it gets to the fifth frame buffer it will capture the next image to the first frame buffer. This will continue until it is stopped by DCAM_IDLE.

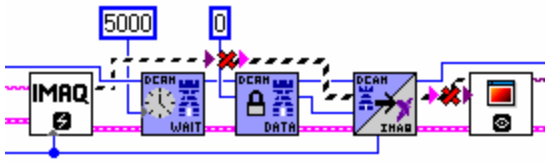
Another difference between a snap and preview capture is the while loop. For each frame that we receive, we would need to process them. The first step in this loop is DCAM_WAIT which is setup up the same as the snap. DCAM_LOCKDATA is now taking -1 for the frame we want to receive instead of 0. -1 means that DCAM_LOCKDATA will get the most recent frame received. We then call DCAM2IMAQ to convert the DCAM buffer to an IMAQ buffer. And then we use IMAQ Display to display the image to the screen. DCAM_UNLOCK is used to unlock the frame so it can be used again by DCAM.

This loop will continue until the user stops the capture by pressing the stop button. In this example we have a stop button provided.

Appendix

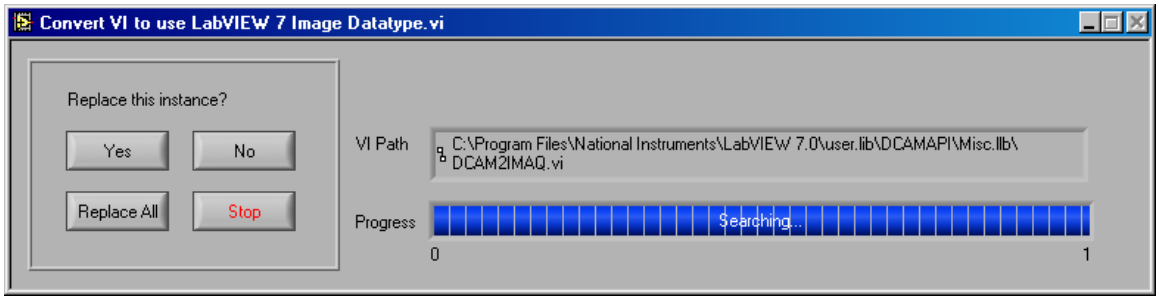
IMAQ Vision 7

National Instruments has updated the IMAQ image type in Vision 7.0. This image type is incompatible with the image type used in the Hamamatsu video capture library. Because of this, the VIs will be unable to run. They will have broken wires similar to this example.



However, Vision 7.0 comes with an upgrade utility that will easily solve this issue. This utility is located in your \National Instruments\Vision\Utility\ folder. Simply run this utility and convert the Hamamatsu Video Capture VIs to update them for use in Vision 7.0.

To update the Hamamatsu Video Capture VIs, you simply need to update DCAM2IMAQ.VI. After selecting this file, the program will begin to search for the old data type and ask you if you want to replace it with the new data type. Select Yes or Replace All to update the VI.



After the file is updated, all other VIs that used it will be able to function correctly. The broken wires associated with the old data type will be corrected and look similar to this example.