



**HR Camera
Software Development Kit
Programmers' Reference Manual**

**DRAFT V 1.0.15
For SDK V 1.0.0.11
24Mar2005**

**PRELIMINARY
BETA USE ONLY
Copyright 2005**

Table of Contents

Scope of this document.....	3
Introduction.....	4
Glossary.....	5
Contents of the SDK.....	6
Distribution and Support.....	7
1.2Distribution.....	7
1.3Support.....	7
Software Licensing.....	8
Supported configurations.....	9
1.4Computer platform.....	9
1.5Operating systems.....	9
1.6Software development environment.....	9
2General system block diagram.....	10
Theory of operation.....	11
Using the HR Library: a simple sequence.....	12
Using the SDK sample code.....	13
Guidelines for robustness and reusability.....	14
Function Reference.....	15
2.1Access to Library, Camera, and Image Parameters.....	15
2.2Control of the HR Library.....	17
2.3Control of a Camera Object.....	19
2.4Control of Image Objects.....	20
Parameter Reference.....	23
2.5 Parameter ID's for Library objects.....	23
2.6Parameter ID's for Camera objects.....	25
2.7Parameter ID's for Image Objects.....	33
Image File Formats.....	37
2.8JPEG.....	37
2.9TIFF.....	37
Lens Control.....	38
2.10Network Interface.....	38
2.11Windows.....	38
2.12Linux.....	38
Revision History.....	39

Scope of this document

This reference document is written for persons designing and writing Host Application software that runs on a computer connected to an Imaginant HR-1100c color digital camera. Details of the camera interface, setup, and control are documented in the following pages. Examples of basic camera operation are included, along with more detailed description of specific features that set the HR-1100c apart from other digital cameras.

Introduction

The Imaginant HR Camera Software Development Kit (SDK) is intended to:

- Simplify integration of Imaginant HRX cameras into a customer's Host Application.
- Hide details of camera protocol and sequencing that are not relevant to an application.
- Provide forward compatibility to Host Applications.
- Handle more than one camera model with the same interface.
- Provide example code that can be used by external developers to test a system.
- Provide example code that can be modified by external developers to their needs.

Glossary

The words and phrases below have specific meaning within this document.

1.1.1 Host Application

The Host Application is the top-level software written by some party other than Imaginant. The Host Application will control the overall sequence of events, usually through a graphical user interface. The Host Application may be written for a special purpose, like machine vision, microscopy, or surveillance.

1.1.2 Host Computer

The whole computer consisting of both hardware and software and includes the Host Application and HR Library.

1.1.3 HR Camera

An HR Camera is a digital camera made by Imaginant, Inc.

1.1.4 HR Library

The HR Camera Library (hereafter called “HR Library”) is a dynamic code library that provides the interface between a Host Application and one or more Imaginant HR cameras. HR Library will transfer information between the Host Application and an HR Camera, will send commands from the Host Application to a camera, and will transfer image data from the HR Camera to the Host Application. On a computer running Microsoft Windows® the HR Library is a file named “HR_Lib.dll”. On a Linux system HR Library file is named “HR_Lib.so”.

1.1.5 SDK

The Imaginant HR Camera Software Development Kit (hereafter called “SDK”) is the complete package of documentation and software described by this document.

Contents of the SDK

HR_SDKProgrammersReferenceManual.pdf This document
HR_Lib.dll The dynamic library that implements the code.
HR_Lib.lib Allows a Host Application to use the dll without dynamic loading.
Header files to be linked into the customer's code.
 HR_Lib.h defines interface functions
 HR_Types.h defines data types
 HR_Errors.h defines error code numbers and meaning
 HR_ParamID.h defines enumerated values for each interface parameter
 HR_Loader.h optionally allows run-time loading of HR_Lib.dll
 HR_Loader.h optionally allows run-time loading of HR_Lib.dll
 HR_LibSignatures.h optionally allows run-time loading of HR_Lib.dll
HR_Loader_Win.c source code to provide run-time loading of HR_Lib.dll
HR_FAQ.pdf Frequently Asked Questions, published more frequently than the Reference Manual.
ReleaseNotes.txt defining new features and known bugs.
Sample source code, with project and make files.

Distribution and Support

1.2 Distribution

The SDK is available as files on a CD-ROM.

After the Beta period is over, files will be available on www.Imaginant.com.

1.3 Support

HR_SDK_ProgrammersReferenceManual.pdf

HR_FAQ.pdf

Email to TechSupport@Imaginant.com

Software Licensing

The "Contents of the SDK" are described in section 5 of this document. Source code of the SDK remains the proprietary intellectual property of Imaginant and is not part of the released or licensed components of the SDK.

Any party that has purchased one or more Imaginant HR Cameras is granted a license to use the SDK.

Any party that has purchased or leased a system that includes one or more Imaginant HR Cameras is granted a license to use the SDK.

External developers are granted a license to use the SDK within their software product and distribute parts of the SDK with their product.

Imaginant holds the copyright of all software in the SDK except portions that are otherwise noted.

A small number of third-party (e.g. Microsoft®) dynamic libraries may need to be used with the HR Camera SDK. External developers will be required to acquire those libraries directly from a Microsoft® CD-ROM or the Microsoft® web site or from a CD-ROM or web site of the manufacturer of the Ethernet interface board used in their Host Computer.

This software is based in part on the work of the Independent JPEG Group, Copyright (C) 1991-1996, Thomas G. Lane. All Rights Reserved.

Supported configurations

1.4 Computer platform

A Pentium® or compatible or higher performance computer.

Minimum 256 MB RAM

Minimum 500 MB Disk space

1.5 Operating systems

Microsoft® Windows 2000 Professional

Microsoft® Windows 2000 Home

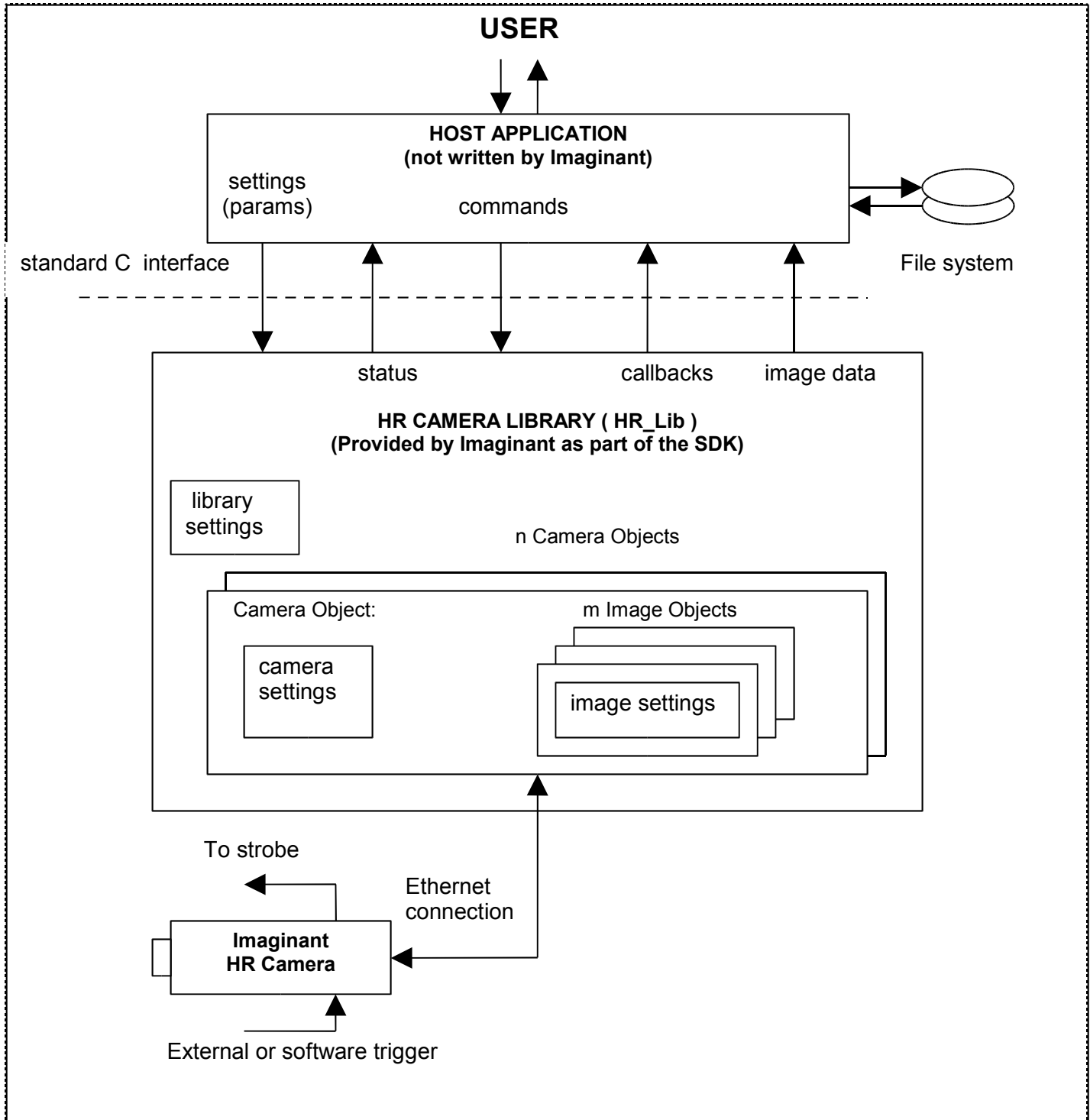
Windows XP Professional

Windows XP Home

1.6 Software development environment

HR_Lib.dll will run with any Host Application designed to use DLL's with a standard C interface. The Sample Code can be used with any development environment using the "C" language and will include project files already made for Microsoft® Developer Studio.

2 General system block diagram



Theory of operation

The HR Camera is connected to the computer using an Ethernet connection, making the camera appear on the network as a simple Ethernet device with three ports. The connection uses a proprietary protocol layer that is handled by the HR Library.

The HR Library model has objects within objects. The Library Object can contain one or more Camera Objects and a Camera Object can contain none, one, or several Image Objects. A unique “handle” identifies each of these various objects. A handle is not a pointer and it should never be incremented or otherwise treated as a mathematical quantity, except that if a handle variable contains zero, it is not a valid handle. Creation of all handles always occurs within the HR Library. Handles are passed up from the HR Library to the Host Application after they are created and then passed back to the HR Library when the Host Application refers to a specific camera or specific image. Every object has some internal settings that can communicate between the Host Application and the object as read-only or read-write enumerated “Params.”

The Host Application needs to initialize the HR Library before communication with an HR Camera. The Host Application can get a list of available cameras from the HR Library. The Host Application then commands the HR Library to initialize one particular camera, creating a Camera Object within the HR Library. The Host Application may then communicate with the camera through that object within the HR Library, including any of the default settings in the camera, such as white balance, file type, ISO, trigger and strobe sequencing, etc.

The HR Camera can be triggered by either an external hardware trigger, a software trigger initiated by the Host Application, or via timed triggers initiating in the HR Camera. As each image is acquired by the HR Camera, it is rendered, is optionally JPEG compressed, and transferred to memory within the Host Computer. As each image arrives at the HR Library, it becomes an Image Object and is assigned a unique HR_ImgHandle. The Host Application can request a list of Image Handles that are currently available within a Camera Object at any time.

The Host Application must “open” an Image Object to work with or communicate with that Image Object. The Host Application may then read, and in some cases modify, some parameters of the Image Object using its Image Handle. Normally, the Host Application will then either write the image data to an image file, render the image data to a display, or analyze the image data. The HR Library does not provide complex image processing such as sharpening or noise filtering, nor does the HR Library provide any image analysis processing. The Host Application should close an Image Object when the application is done with the Image Object.

The Host Application can command the HR Library to delete Image Objects as it needs to, which frees up all memory within the HR Library associated with that image, completely deleting that image from the HR Library.

Using the HR Library: a simple sequence

Steps within the Host Application's code:

- Dynamically Load HR_Lib.dll
- Open the HR_Library.
- Get list of available cameras from HR_Library
- Connect to ("Open") one camera through HR_Library
- Set parameters into the camera through HR_Library
- Do this any number of times:
 - Command "HR_TriggerCamera" through HR Library or hardware trigger
 - Poll the camera through HR_Library to wait for an image
 - Open the image within HR_Library
 - Wait for image transfer complete
 - Get image data from HR_Library to the Host Application
 - Write the image data as a file to the file system through the OS
 - Close the image within HR_Library
 - Delete the image within HR_Library
- Close the camera through HR_Library
- Close HR_Library
- Release the dynamic load of HR_Lib.dll

Using the SDK sample code

The Sample Code provided as part of the SDK can be used for

- Testing system and camera operation
- Examples of controlling specific features of the HR Library and cameras
- Providing a cut-and-paste source of working code for external developers.

The Sample Code provided as part of the SDK consists of

- Source code
- Header files
- Project definition files
- Executable code

Documentation for each piece of source code can be found within its documents.

We strongly recommend the Host Application use Dynamic Loading of HR_Lib.dll as described in a section below entitled "Guidelines for robustness and reusability."

Guidelines for robustness and reusability

The HR Library has been designed and implemented so additional camera models with different imager height and width and other different properties can work with the same interface. The guidelines in this section can help you design and code an application that can seamlessly work with several different camera models.

The HR Library has been designed and implemented so a newer version of the HR_Lib dynamic library can be used by an older version of Host Application with no change to the Host Application and no need to rebuild the Host Application. Obviously, features in the newer version of the dynamic library cannot be used until coded into the Host Application.

Host Application software should always test the HR_StatusType value returned by every call to a function in the HR Library and properly respond to exceptions.

Host Application software should not hard-code values that may change for different camera models or even for different image formats produced by one camera. Rather, values such as image width and image height should always be read from the HR Library using HR_GetParam(...).

Host Application software should be built with files HR_Types.h, HR_Errors.h, HR_ParamID.h, and HR_Lib.h, “#included”. Identifiers from those files should be used in Host Application code, rather than hard-coded numeric values.

Host Application software should always delete an Image Object with HR_DeletelImage() when it is finished using the image or has decided to ignore the image. Doing so will cause the HR Library to release memory it had reserved “under the hood.” If the Host Application keeps a copy of an image handle in an array or member variable, the Host Application should assign 0 to that handle after commanding the HR Library to delete the image associated with the handle.

The HR Library zip file contains files that allow the Host Application developer to either
Dynamically load the HR_Lib.DLL file at run time using the HR_Loader files or
Link your code with HR_Lib.lib to cause a static link to HR_Lib.DLL

Although both methods work, we strongly advise using the Dynamic Loading method. Doing so makes the HR_FunctionPtrs struct act similar to an object within the Host Application, allows the Host Application to degrade or exit gracefully if the HR_Lib.dll is not found, and would allow multiple DLL's with the same API to be used simultaneously with the same code within the Host Application. Files HR_Loader.h, HR_Lib_signatures.h, and HR_Loader_Win.c can be used within the Host Application software to provide dynamic loading. The Sample code files use dynamic loading to help demonstrate how to use dynamic loading.

When the Host Application uses dynamic loading of HR_Lib.dll the file HR_Lib.lib should not linked with the Host Application code. Attempting to do so will cause a “stack overflow” error at run time, but no other warning.

Function Reference

NOTE: This User Guide is a preliminary document. All definitions of interface functions are subject to change as long as this statement is in the User Manual.

2.1 Access to Library, Camera, and Image Parameters

2.1.1 HR_GetParamInfo

```
HR_StatusType HR_GetParamInfo
(
    HR_BaseHandle      inHandle,          // a handle to a lib, cam, or img object
    HR_ParamIdEnumType inID,             // enumerated ID of a parameter
    HR_InfoType*       outInfo           // all the param info is in this struct
);
```

HR_GetParamInfo(...) allows the Host Application to determine information about any parameter of the HR Library, an open camera, or an open image. Note that this function does not provide the actual value of a parameter, just information about the parameter. This function does not need to be called for simple parameters, such as a single integer, but is especially useful to determine the size of an array or the size of a dynamic list, such as the number of available images or the size of a string.

HR_GetParamInfo(...) can be used to determine:

- The existence and validity of a parameter (an error return indicates non-valid)
- The data type of a parameter.
- The read/write access of a parameter.
- The count of elements of a parameter.
- The count of bytes of a parameter (may be different from the count of elements).

NOTE: For the special case of an HR_StringType parameter, outInfo.elementCount and outInfo.byteCount will be the same and contain the count of characters including the terminating zero. That will allow the Host Application to allocate the number of bytes in the outInfo.byteCount. This requirement will cause an empty string to have a byteCount and elementCount of one (the terminating zero). A call to HR_GetParam(...) for a parameter of HR_StringType will always put a terminating zero on the string copied into the Host Application's allocated variable or buffer. If the Host Application has requested one character too few, the HR Library will overwrite the last character with a terminating zero. That behavior is more desirable than causing a crash sometime later within the Host Application.

The Host Application can tell the difference between an empty string and a non-existent string. If the particular string parameter your Host Application is requesting information about does not exist for the specified Object, the call to HR_GetParamInfo(...) will return an error status of HR_ERR_PID_NOT_VALID. If the string is present but empty, the status return value will be HR_OK but outInfo.byteCount will contain the number 1 and the first element of the string will be a terminating zero.

Both the count of elements and byte count are provided to allow the Host Application to use whichever is more convenient. The byte count can be used directly to allocate a buffer.

When HR_GetParamInfo(...) returns, the outInfo.dataType field will contain the enum value can be passed as the inType argument to HR_SetParam(...) and may also provide useful information while debugging code in the Host Application.

The application does not need to call HR_GetParamInfo(...) for every parameter. Most parameter values have a fixed count and a constant data type. Those constants can be coded into the application's statements that call HR_GetParam(...) or HR_SetParam(...).

2.1.2 HR_GetParam

```
HR_StatusType HR_GetParam
```

```
(
    HR_BaseHandle    inHandle,          // handle to a lib, cam, or img object
    HR_ParamIdType   inID,              // enumerated ID number of a parameter
    HR_Int32Type     inCount,           // the count of outpValue in elements
    HR_Int32Type*    outAvailCount,     // count of elements (not bytes) available
    void*            outpValue          // SDK will copy data here
);
```

HR_GetParam(...) allows the Host Application to get the value of a parameter in the HR Library, an open camera object, or an open image object, whether the parameters is a simple integer value, an array, a data structure, or an array of structures.

The Host Application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the Host Application. The HR Library will copy the value (or struct or array) into the Host Application's memory space.

The Host Application has control of how many pieces of information will be copied by the HR Library by setting the value of the inCount argument. The memory the Host Application allocated for outpValue must be large enough to hold inCount number of elements.

When HR_GetParam(...) returns with HR_OK, outAvailCount will report the number of elements that were available, which could be zero, less than inCount, the same as inCount, or greater than inCount.

The Host Application can control the inCount argument to get a subset of a list or array.

For parameters that may have a dynamic count, such as the element count of a string or array, the Host Application will normally call HR_GetParamInfo(...) to determine the element count and byte count of a parameter, then allocate the needed memory, and then call HR_GetParam(...).

For the special case of an HR_StringType parameter see the special note for the HR_GetParamInfo(...) function.

2.1.3 HR_SetParam

```
HR_LIB_API HR_StatusType HR_SetParam
(
    HR_BaseHandle    inHandle,          // handle to a lib, cam, or img object
    HR_ParamIdType   inID,              // enumerated ID of a parameter
    HR_TypeType      inType,            // data type of the param
    HR_Int32Type     inCount,           // count of elements (not bytes)
    void*            inpValue           // points to a value or array
);
```

HR_SetParam(...) allows the Host Application to set the value of a parameter into the HR Library, an open camera object, or an open image object, whether the parameter is a simple integer value, an array, a data structure, or an array of structures.

The Host Application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the Host Application. The HR Library will copy the value (or struct or array) from the Host Application's memory space into the object within the HR Library.

The Host Application has control of how many pieces of information will be copied into the HR Library by setting the value of the inElementCount argument to HR_SetParam(...). If the Host Application attempts to set a count of elements larger than the HR Library is expecting, the function will not perform any action except to return the error message HR_ERR_PARAM_COUNT_TOO_BIG.

Some parameters provide read-write (HR_ACCESS_RW) access but other parameters provide read-only (HR_ACCESS_RO) access. If the Host Application attempts HR_SetParam(...) on a parameter that has HR_ACCESS_RO (read-only) access, HR Library will perform no action except to return the error message HR_ERR_PARAM_READ_ONLY.

One of the arguments to `HR_SetParam(...)` is the data type of the parameter. If the HR Library detects the data type as not matching the data type associated with the parameter identifier enum the HR Library will perform no action except to return the error message `HR_ERR_WRONG_DATA_TYPE`.

The access mode of a parameter can be determined ahead of time by the Host Application by calling `HR_GetParamInfo(...)` and is documented for each parameter in `HR_Types.h`.

NOTE: For the special case of an `HR_StringType` parameter `HR_SetParam(...)` expects the last character to be a terminating zero and the `inElementCount` argument should include the terminating zero. (`strlen(s) + 1`) If the last character is not the value zero, the call to `HR_SetParam(...)` will return `HR_ERR_STRING_NOT_TERMINATED`.

2.1.4 HR_GetErrorString

```
char* HR_GetErrorString
(
    HR_StatusType inErrorNumber // status returned by some other HR Lib function
);
```

`HR_GetErrorString` decodes a number into a zero-terminated string representation of the `inErrorNumber` and the enum as defined in `HR_Errors.h`.

For example, if the value 100 is the input argument, the string created will be
" 101 HR_ERR_MEM_ALLOC_FAILED "

`HR_GetErrorString` always returns a pointer to a string, even if the `inErrorNumber` argument is "HR_OK". In that case, the string will be " 0 HR_OK" If the `inErrorNumber` is not valid, e.g. 4321, the string will display:
"4321 IS NOT A VALID HR_Lib ERROR NUMBER"

To allow the application more flexibility, the string does not have "\n" at the end.

The returned `char*` always points to the same buffer internal to the HR Library, which will be overwritten the next time `HR_GetErrorString` is called. Therefore, the application should either display the string immediately or make a copy of the string for later display.

Example:

```
status = HR_SomeFunction(...)
if ( status != HR_OK )
    printf("HR_Lib ERROR %s\n", HR_GetErrorString(status) );
```

2.2 Control of the HR Library

2.2.1 HR_OpenLibrary

```
HR_StatusType HR_OpenLibrary
(
    HR_Int32Type inOptionBits, // Unused bits must be zero
    HR_LibHandle* outLibHandle // handle needed for later calls
);
```

`HR_OpenLibrary(...)` checks for the existence of the `HR_Library`, establishes a virtual connection to it, allows control of some options, and returns a handle to the Library Object which will be used by the Host Software as the `inObjectHandle` for some subsequent calls to the HR Library.

`HR_OpenLibrary(...)` allocates memory, may initiate processing threads and other resources, so `HR_CloseLibrary(...)` should be used before a Host Application closes in order to release the memory and other resources.

The `inOptionBits` parameter to `HR_OpenLibrary(...)` allows 32 on/off control options. The state of these options can not be modified after the HR Library is opened. The default state of all options will be specified as zero, so `inOptionBits = 0` is always a valid value.

Additional options may be added in HR Library versions after 1.0. Therefore a Host Application must have all unused bits set to zero. If the HR Library detects any option bits are set to one that are not allowed, `HR_OpenLibrary(...)` will return `HR_ERR_INVALID_OPTION_BITS`.

2.2.2 HR_Poll

```
HR_StatusType HR_Poll
(
    HR_BaseHandle inHandle,          // A Lib, Cam, or Img Handle
    HR_Int32Type  inEventGroup,      // Unused bits must be zero
    HR_Int32Type* outEventFlags      // handle needed for later calls
);
```

During operation of the camera and HR Library events or changes of state occur in the library which are asynchronous to the Host Application need to be communicated from the library to the Host Application. For example, if an image capture is initiated using the camera's external trigger connector, the image is automatically uploaded via Ethernet to a buffer in the library running on the Host Computer. It is necessary for the HR Library to notify the host application that the image needs to be read and/or deleted.

The simplest way for the Host Application to accomplish this notification is through a polling call. Periodically, the Host Application can call `HR_Poll()` to determine which events have happened since the last polling call. Because of the large number of possible events, they are collected into event groups. This allows a developer to request which event groups have received events, and process those event groups separately.

A call to `HR_Poll()` takes three arguments, an API object handle, an enum value indicating the event group of interest, and a pointer to an `HR_Int32Type` (`&outEventFlags`) to return the event codes. After `HR_Poll()` returns `HR_OK`, the caller can check for specific events via a bitwise AND with enum values such as `HR_EVENT_CAM_TRANSFER_NEW_COMPLETE_IMAGE_AVAILABLE`.

When a specific event group is polled (commands ending in “_EVENTS”), all of the event codes in that group are read and cleared inside the HR Library. Clearing the events during each call avoids the need for a second call to reset the flags, but it also makes it necessary for the caller to handle all events retrieved in that group at the same time, using the value of `outEventFlags` that was returned by one call.

Since polling each event group with separate calls could be a burden on the Host Application code, the HR Library has the ability to do a global check for which set of event groups have flags set. Calling `HR_Poll()` with any commands that end in “_GROUPS_AVAILABLE” returns in `outEventFlags` a flag set indicating which event groups have received events since that event group was last polled. Note that these global calls do not clear the internal event bits.

For example, a camera application can poll using `HR_POLL_CAM_GROUPS_AVAILABLE` which will return a set of flags. If the `HR_EVENT_CAM_TRANSFER_EVENTS` bit is set, this indicates that at least one event flag in the set of events `HR_POLL_CAM_TRANSFER_EVENTS`. The group available flag `HR_EVENT_CAM_TRANSFER_EVENTS` will remain set until the user polls the camera for camera transfer events.

2.2.3 HR_CloseLibrary

```
HR_StatusType HR_CloseLibrary
(
```

```
HR_LibHandle inLibHandle // the handle returned by HR_OpenLibrary
);
```

HR_OpenLibrary(...) allocates memory, may initiate processing threads and other resources, so HR_CloseLibrary(...) should be used before a Host Application closes in order to release the memory and other resources.

2.3 Control of a Camera Object

A Camera Object is the software representation of a physical HR Camera, the connection to the camera, and settings within the object that affect how the camera collects or processes images. When the Host Application sets parameters to a Camera Object, those parameters are stored within the Camera object within the HR Library and in some cases get sent to the physical HR Camera. Likewise, the Host Application can read parameters and status of a physical HR Camera by reading parameters from a Camera object.

The HR Library Object can contain zero, one, or a number of Camera Objects.

A Camera Object can contain zero, one, or a number of Image Objects.

To prevent system lock-up or crashing from a Camera Object attempting to store more Image Objects than the Host Computer's memory can hold, the Camera Object provides options for what it should do when either the maximum number of images is exceeded or no more memory is available, including "HR_ACTION_DELETE_OLDEST_IMAGE" or "HR_ACTION_IGNORE_NEW_IMAGES".

A Camera Object has the parameter HR_PARAM_CAM_MAX_IMAGE_COUNT to allow the Host Application to set that value to something other than the default.

2.3.1 HR_GetCameraList(...)

```
HR_StatusType HR_GetCameraList
(
    HR_LibHandle inLibHandle,    // must be a valid Library handle
    HR_Int32Type inBufferCount,  // must be greater than 0
    HR_Int32Type* outAvailCount, // number of cameras available
    HR_CamHandle* outCameraList  // must be allocated at least inBufferCount
);
```

The HR Library can be simultaneously connected to several cameras. HR_GetCameraList (...) is used by the Host Application to get the list of Camera Handles of cameras currently connected to the Host Computer. The number of connected cameras can change many times during any session, if a camera becomes disconnected or more cameras become physically connected to the Host Computer.

The Host Application must allocate a buffer to receive the list of Camera Handles and must set argument inBufferCount to indicate how big the allocation was. The count of cameras found will always be returned in *outAvailCount, whether *outAvailCount is less than, the same as, or greater than the value of inBufferCount.

A mismatch of *outAvailCount and inBufferCount is not an error situation.

If no Camera Objects are found, HR_GetCameraList(...) will return HR_OK but the outAvailCount value will contain zero to indicate that condition.

If the number of Camera Objects available is greater than inBufferCount, that is not an error condition: HR_GetCameraList (...) will return HR_OK. The outAvailCount value will contain the actual number of cameras currently available even if that number is larger than inBufferCount. If a Host Application can only work with one camera at a time, the application can set inBufferCount to one, work with that camera, then close it, then call HR_GetCameraList (...) again as many times as it needs to.

A Camera Handle will always be a unique number for a given HR Library session. If two different cameras are connected, the HR Library will assign unique numbers for the handles. However, closing the HR Library and reopening it will invalidate any Camera Handles or Image Handles that were open by that session of the HR Library.

If a specific camera had a successful connect to the HR Library, but then gets disconnected and then connected again, the HR Library will reconnect to that camera using the same handle. While the camera is disconnected, the camera parameter `HR_PID_CAM_STATUS` will indicate the changed state. Attempts by the Host Application to communicate with the camera through the HR Library while the camera is disconnected will cause error return values.

2.3.2 HR_OpenCamera

```
HR_StatusType HR_OpenCamera
(
    HR_CamHandle inCameraHandle    // must be a valid handle
);
```

`HR_OpenCamera(...)` causes the HR Library to establish a connection to a physical HR Camera.

`HR_OpenCamera (...)` allocates memory and other resources, so `HR_CloseCamera(...)` should be called by a Host Application before it closes the HR Library in order to release the memory and other resources.

2.3.3 HR_CloseCamera

```
HR_StatusType HR_CloseCamera
(
    HR_CamHandle inCameraHandle    // must be a valid open camera handle
);
```

`HR_CloseCamera(...)` causes the HR Library to release a connection to a physical HR Camera and releases all resources associated with the Camera Object within the HR Library. `HR_CloseCamera(...)` sends a “reset” command to the camera.

`HR_CloseCamera (...)` should be called by a Host Application before it closes the HR Library in order to release the memory and other resources.

2.3.4 HR_TriggerCamera

```
HR_StatusType HR_TriggerCamera
(
    HR_CamHandle inCameraHandle    // must be a valid open camera handle
);
```

`HR_TriggerCamera(...)` causes the camera to begin a trigger sequence which can be one or more image exposures depending on the settings of camera parameters.

The parameter `HR_PID_CAM_TRIGGER_ENABLE` must be true before calling `HR_TriggerCamera(...)`. If `HR_PID_CAM_TRIGGER_ENABLE` is false, a call to `HR_TriggerCamera(...)` will return `HR_ERR_CAM_TRIGGER_IS_DISABLED`.

2.4 Control of Image Objects

A Camera Object may contain zero, one, or several Image Objects. An Image object is both a representation of an image and contains the memory allocated to an image. That memory allocation is within the memory space of the HR Library.

The HR Library constructs an Image Object when a physical HR Camera informs the HR Library that an image transfer has started. This Image Object construction takes place completely “under the hood” and does not require any action by the Host Application layer. The existence of an Image Object does not necessarily mean the whole image is in memory on the Host Computer.

An Image Handle is a number used between the host Application and the HR Library to identify a specific instance of an Image Object within a Camera Object. It has no numerical relation to the “Image Number”. It is not a pointer. It can not be incremented or decremented or otherwise treated as a mathematical quantity except that an Image Handle value of zero is not a valid image.

After opening an Image Object, the Host Application can and should determine the state of image transfer by calling `HR_GetParam(myCameraHandle, HR_PARAM_IMG_PERCENT_DONE...)`

The HR Library assigns an Image Handle when an Image Object is created by a notification from the HR Camera that an image is coming.

2.4.1 HR_GetImageList(...)

```
HR_StatusType HR_GetImageList
(
    HR_BaseHandle inHandle,          // can be the Library or one specific camera
    HR_Int32Type  inBufferCount,     // elements the app. allocated for outImageList
    HR_Int32Type* outAvailCount,     // number of images available
    HR_ImgHandle* outImageList       // allocation must match inBufferCount
);
```

A Camera Object can contain none, one, or several Image Objects at one time. `HR_GetImageList(...)` is used by the Host Application to get the list of Image Handles within the specified Camera Object. The number of available images can change many times during any session, if the Host Application deletes any images or a camera sends new images to the Host Computer.

The Host Application must allocate a buffer to receive the list of Image Handles and must set argument `inBufferCount` to indicate how big the allocation was. The count of images within the Camera Object will always be returned in `*outAvailCount`, whether `*outAvailCount` is less than, the same as, or greater than the value of `inBufferCount`.

A mismatch of `*outAvailCount` and `inBufferCount` is not an error situation.

If no Image Objects are found, `HR_GetImageList(...)` will return `HR_OK` but the `outAvailCount` value will contain zero to indicate that condition.

If the number of Image Objects available is greater than `inBufferCount`, that is not an error condition: `HR_GetImageList(...)` will return `HR_OK`. The `outAvailCount` value will contain the actual number of images currently available even if that number is larger than `inBufferCount`. If a Host Application can only work with one image at a time, the application can set `inBufferCount` to one, work with that first image, then delete it, then call `HR_GetImageList(...)` again as many times as it needs to.

An Image Handle will always be a unique number for a given HR Library session. If two different cameras are connected, the HR Library will never assign the same numeric value for an Image Handle to one image from each camera. However, closing the HR Library and reopening it will invalidate any Camera Handles or Image Handles that were open by that session of the HR Library.

There are two ways for the Host Application to get an Image Handle.

- If callbacks are enabled, the `HR_EVENT_NEW_IMAGE` carries the handle of the new image.
- The Host Application can perform `HR_GetImageList(...)` at any time to get a list of handles of Image Objects that currently exist within the Camera Object.

2.4.2 HR_OpenImage

```
HR_LIB_API HR_StatusType HR_OpenImage
(
    HR_ImgHandle inImageHandle       // must be a valid image handle
);
```

An Image Object must already exist within a Camera Object before the Host Application calls this function, passing an Image Handle described above. Think of this as similar to opening a file you plan to read from. Once open, parameters of the Image Object may be read (and some written) by the Host Application.

`HR_OpenImage(...)` should be matched with a call to `HR_CloseImage(...)`. `HR_DeleteImage(...)` will also be needed to completely delete the image.

2.4.3 HR_CloseImage

```
HR_StatusType HR_CloseImage
(
    HR_ImgHandle inImageHandle    // must be a valid open image handle
);
```

HR_CloseImage(...) releases some of the memory and other resources associated with an Image Object that was opened using HR_OpenImage() but it does not delete the actual image data that came from the HR Camera. The image data is retained so the image can be opened again later. A “closed” Image Object may still have several megabytes of memory allocated to it. HR_DeletelImage(...) must be called to completely remove all resources allocated to an Image Object.

2.4.4 HR_DeletelImage

```
HR_StatusType HR_DeleteImage
(
    HR_ImgHandle inImageHandle    // must be a valid handle
);
```

HR_DeletelImage(...) releases all of the memory and other resources associated with an Image Object, including the actual image data. After calling HR_DeletelImage(...) the image data and all parameters will be completely gone. The Host Application must never attempt another operation with that Image Object Handle. To do so will cause a return status of HR_ERR_IMG_INVALID_HANDLE.

In some cases, the HR Camera and HR Library may have acquired images the Host Application does not need and therefore are just ignored by the Host Application. The Host Application must call HR_DeletelImage(...) for all Image Objects, whether it opened them or not. Failure to do so may cause the Host Application to run out of memory or cause the Host Application to run very slowly as the operating system begins to use virtual memory.

Parameter Reference

Parameters, also known as settings, of the Library Object, a Camera Object, or an Image Object are accessed using the `HR_GetParamInfo()`, `HR_GetParam()`, and `HR_SetParam()` functions. Which particular parameter of each object is accessed is selected by the `inParamID` argument to those functions. The `inParamID` argument must be a valid parameter enum value from `HR_ParamID.h`.

Applications should `#include "HR_ParamID.h"`, and use the identifier for the desired parameter. Never hard-code an integer constant.

2.4.5 HR_PID_UNKNOWN

This enumeration is always zero and should cause any of the three “param” functions in the API to return `HR_ERR_PID_NOT_VALID`. Any other number that is not a valid parameter identifier enumeration should also return the error status of `HR_ERR_PID_NOT_VALID` or `HR_ERR_LIB_PID_OUT_OF_RANGE`.

2.4.6 HR_PID_ALL_HANDLE_TYPE

Data Type	Count	Access	Object
<code>HR_TYPE_INT32</code>	1	Read-Only	Library, Camera, Image

`HR_PID_ALL_HANDLE_TYPE` works for all three object types. The parameter value will be an enum value of `HR_ApiObjectEnumType` which allows the application to verify the type of API object.

2.5 Parameter ID's for Library objects

2.5.1 HR_PID_LIB_VERSION

Data Type	Count	Access	Object
<code>HR_TYPE_INT32</code>	4	Read-Only	Library

Returns the four numbers of the version resource of the DLL.

2.5.2 HR_PID_LIB_NAME

Data Type	Count	Access	Object
<code>HR_TYPE_STRING</code>	Variable	Read-Only	Library

Returns an ASCII string identifying the Library. The application can determine the count of characters by calling `HR_GetParamInfo(...)`. An empty string will have a count of 1 for the terminating zero of a standard C string.

2.5.3 HR_PID_LIB_OPEN_OPTIONS

Data Type	Count	Access	Object
<code>HR_TYPE_INT32</code>	1	Read-Only	Library

Returns the `inOptionBits` argument that was passed into `HR_OpenLibrary(...)`. This is provided in case the Library Object was opened by a different thread.

2.5.4 HR_PID_LIB_START_DATE_TIME

Data Type	Count	Access	Object
<code>HR_TYPE_STRING</code>	20	Read-Only	Image

This parameter reports the date and time the library process started in EXIF format:
YYYY:MM:DD HH:MM:SS

The date-time string comes from the Operating System of the Host Computer. The EXIF format is useful because it is both human readable and can be sorted using `strcmp()` or STL string class compares. The count of 20 includes the terminating zero, consistent with the `HR_TYPE_STRING`, TIFF, and EXIF definitions.

2.5.5 HR_PID_LIB_MEMORY_ALLOWED_MB

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Library

BACKGROUND: HR Cameras can be configured to trigger image captures from software triggers, external hardware triggers, or continuous Preview captures. In external trigger or Preview mode the HR Cameras can continually capture and send images via the Ethernet. The HR Library normally continues to accept images sent by the HR Camera(s) without interaction with a Host Application. If the Host Application does not send commands to the HR Library to delete images the HR Library could allocate all of the available memory in the Host Computer, causing a crash, a failure message, or a dramatic slowdown while using virtual memory.

The HR_PID_LIB_MEMORY_ALLOWED_MB parameter allows the Host Application to control how much memory (in MegaBytes, 1024x1024) in the computer can be used by all Image Objects within all connected Camera Objects.

When an image begins to arrive from an HR Camera but allocating the memory it needs would cause the HR Library to allocate more memory than the Host Application has specified by the HR_PID_LIB_MEMORY_ALLOWED_MB parameter, the HR Library will:

- Accept the image from the camera, but discard it.
- Continue accepting images from the camera, but discard them.
- The Image Object handle will not appear in HR_GetImageList(...)
- Cause a Camera Callback (if enabled) HR_EVENT_IMG_ALLOWED_MEMORY_EXCEEDED
- Cause an HR_Poll() command to the Camera Object to indicate the problem.
- Cause HR_ERR_CAM_ALLOWED_MEMORY_EXCEEDED to be returned for
 - Calls to HR_SetParam(HR_PID_CAM_RUN_PREVIEW, true)
 - Calls to HR_TriggerCamera(...)

A similar potential problem will be detected if the SDK senses that accepting an incoming image would cause the number of bytes of memory allocated to image buffers exceeds 90% of the currently available physical memory in the computer. The HR Library will behavior similar to that described above, except:

- Cause a Camera Callback (if enabled) HR_EVENT_IMG_PHYSICAL_MEMORY_EXCEEDED
- Cause an HR_Poll() command to the Camera Object to indicate the problem.
- Cause HR_ERR_CAM_PHYSICAL_MEMORY_EXCEEDED to be returned for
 - Calls to HR_SetParam(HR_PID_CAM_RUN_PREVIEW, true)
 - Calls to HR_TriggerCamera(...)

To clear the problem, the Host Application can delete one or more Image Objects. The Host Application could also set a higher value for HR_PID_LIB_MEMORY_ALLOWED_MB but should not exceed the amount of physical memory the application wants to allow the HR Library to use.

The buffer allocation within the HR Library for each JPEG image is larger than actually needed, so to minimize the amount of unused allocated memory, the Host Application should copy the JPEG image data into it's own memory space and then close and delete the Image Object in the HR Library.

The units for HR_PID_LIB_MEMORY_ALLOWED_MB is MegaBytes (1024x1024). Only data for image buffers is counted, so the HR Library will actually allocate more memory, several KB per image.

When the HR_Library is initialized, the value of the HR_PID_LIB_MEMORY_ALLOWED_MB parameter will be 50% of the physical memory that is free at that time. The amount of free physical memory depends on all the applications and services running on the Host Computer at that moment. The value should not be expected to remain stable and should not be considered to have extreme accuracy.

2.5.6 HR_PID_LIB_MEMORY_USED_MB

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Library

HR_PID_LIB_MEMORY_USED_MB works with the HR_PID_LIB_MEMORY_ALLOWED_MB parameter.

HR_PID_LIB_MEMORY_USED_MB reports in rounded-up MegaBytes the amount of memory currently allocated to image buffers within the HR Library.

The units for The HR_PID_LIB_MEMORY_USED_MB is MegaBytes (1024x1024). Only data for image buffers is reported here, so the HR Library will actually allocate more memory, several KB per image.

2.6 Parameter ID's for Camera objects

HR_PID_CAM_STATUS

2.6.1 HR_PID_CAM_STATE

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

HR_PID_CAM_STATE reports the current status of the camera using HR_CamStateEnumType. The enum value HR_CAM_STATE_IDLE indicates the camera is running, waiting for a trigger.

2.6.2 HR_PID_CAM_RUN_PREVIEW

Data Type	Count	Access	Object
HR_TYPE_BOOL	1	Read-Write	Camera

When commanded to true, the camera will continuously capture preview images and send them to the HR_Library. To turn preview mode off, the application should set this value false.

The format of a preview image is not affected by and does not affect HR_PID_CAM_CAPTURE_FILE_TYPE.

The parameter HR_PID_CAM_TRIGGER_ENABLE must be true before calling HR_SetParam(camHandle, HR_PID_CAM_RUN_PREVIEW, &true). If HR_PID_CAM_TRIGGER_ENABLE is false, the attempt to turn on preview mode will return HR_ERR_CAM_TRIGGER_IS_DISABLED.

The Host Application should not call HR_SetParam(camHandle, HR_PID_CAM_TRIGGER_ENABLE, &false) while parameter HR_PID_CAM_RUN_PREVIEW is true (running). If so, the HR_SetParam() function will return HR_ERR_CAM_RUN_PREVIEW_IS_ON. No change will be made to the state of the HR Library or the camera.

2.6.3 HR_PID_CAM_CAPTURE_FILE_TYPE

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Write	Camera

This value specifies which image file type the camera will acquire using the HR_ImgDataFormatEnumType. This value does not affect the format of preview images.

2.6.4 HR_PID_CAM_JPEG_QUALITY

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Write	Camera

This value controls the tradeoff between JPEG image quality and compression ratio. This uses the scaling mechanism of the Independent JPEG Group (IJG). The value can be any integer from 1 to 100, inclusive. Some examples are shown below:

Numeric Quality Value	Image Quality	File Byte Count
1	Very Low	Very Small
25	Low	Small
50	Medium	Medium
75	High	Large
100	Highest	Very Large

2.6.5 HR_PID_CAM_CHANS_PER_PIXEL

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

For a color camera, this count of channels will always be 3. For a monochrome camera this will be 1.

2.6.6 HR_PID_CAM_BYTES_PER_CHAN

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

For the camera has 8 or fewer bits per channel, this value will be 1. If the camera has more than 8 bits per channel, this value will be 2. This value is not affected by HR_PID_CAM_CAPTURE_FILE_TYPE. If the camera is set to acquire JPEG images, the final JPEG data will actually have 1 byte per channel.

2.6.7 HR_PID_CAM_NAME

Data Type	Count	Access	Object
HR_TYPE_STRING	Variable	Read-Write	Camera

This ASCII string is writable by the Host Application and gets copied to the image object and to the header of the image at the time of image acquisition. This allows tracking of the source of the image with a text string that has relevance to the Host Application, which can report the location of the camera or any other information that can be put into an ASCII string.

2.6.8 HR_PID_CAM_CONN_DATE_TIME

Data Type	Count	Access	Object
HR_TYPE_STRING	20	Read-Only	Image

This parameter reports the date and time the camera was connected in EXIF format:
YYYY:MM:DD HH:MM:SS

The date-time string comes from the Operating System of the Host Computer. The EXIF format is useful because it is both human readable and can be sorted using strcmp() or STL string class compares. The count of 20 includes the terminating zero, consistent with the HR_TYPE_STRING, TIFF, and EXIF definitions.

2.6.9 HR_PID_CAM_CAMERA_MAKE

Data Type	Count	Access	Object
HR_TYPE_STRING	Variable	Read-Only	Camera

This ASCII string has the spelling of the camera manufacturer: "Imaginant" The application can determine the count of characters by calling HR_GetParamInfo(...). An empty string will have a count of 1 for the terminating zero of a standard C string.

2.6.10 HR_PID_CAM_CAMERA_MODEL

Data Type	Count	Access	Object
HR_TYPE_STRING	Variable	Read-Only	Camera

This ASCII string has the spelling of the camera model. The application can determine the count of characters by calling HR_GetParamInfo(...). An empty string will have a count of 1 for the terminating zero of a standard C string.

2.6.11 HR_PID_CAM_FIRMWARE_VERSION

Data Type	Count	Access	Object
HR_TYPE_STRING	Variable	Read-Only	Camera

This ASCII string contains the version number string of the firmware in the camera. The application can determine the count of characters by calling HR_GetParamInfo(...). An empty string will have a count of 1 for the terminating zero of a standard C string.

2.6.12 HR_PID_CAM_SERIAL_NUMBER

Data Type	Count	Access	Object
HR_TYPE_STRING	Variable	Read-Only	Camera

This ASCII string contains the serial number of the camera. The application can determine the count of characters by calling HR_GetParamInfo(...). An empty string will have a count of 1 for the terminating zero of a standard C string.

2.6.13 HR_PID_CAM_WHITE_BALANCE

Data Type	Count	Access	Object
HR_TYPE_FLOAT	3	Read-Write	Camera

This does a get or set of the RGB white balance triplet the camera will apply to images.

2.6.14 HR_PID_CAM_WHITE_BALANCE_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

Provides the minimum number allowed for any of the R,G,B values of white balance.

2.6.15 HR_PID_CAM_WHITE_BALANCE_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read- Only	Camera

Provides the maximum number allowed for any of the R,G,B values of white balance.

2.6.16 HR_PID_CAM_COLOR_CORRECTION_MATRIX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	9	Read-Write	Camera

This value is a 3 by 3 matrix that the camera will apply to image data to convert image data from Camera RGB to rendered sRGB while creating JPEG images.

2.6.17 HR_PID_CAM_JPEG_ROI_RECT

Data Type	Count	Access	Object
HR_TYPE_RECT	1	Read-Write	Camera

This param allows the Host Application to control the Region Of Interest to be applied to JPEG images if the camera is set to capture JPEG images.

2.6.18 HR_PID_CAM_JPEG_WIDTH

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the width in pixels of a non-cropped JPEG image that will be captured if the camera is set to capture non-cropped JPEG images.

2.6.19 HR_PID_CAM_JPEG_HEIGHT

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the height in scan lines of a non-cropped JPEG image that will be captured if the camera is set to capture non-cropped JPEG images.

2.6.20 HR_PID_CAM_PREVIEW_ROI_RECT

Data Type	Count	Access	Object
HR_TYPE_RECT	1	Read-Write	Camera

This param allows the Host Application to control the Region Of Interest to be applied to preview images if the camera is set to collect preview images.

2.6.21 HR_PID_CAM_PREVIEW_WIDTH

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the width in pixels of a non-cropped preview image that will be captured if the camera is set to capture preview images.

2.6.22 HR_PID_CAM_PREVIEW_HEIGHT

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the height in scan lines of a non-cropped preview image that will be captured when the camera is set to capture preview images.

2.6.23 HR_PID_CAM_FOCUS_POSITION

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Write	Camera

This controls the focus motor of the lens and reflects the absolute position of the lens in steps of the focus motor. The lens is at closest focus when focus position is set to the value read from HR_PID_CAM_FOCUS_POSITION_MIN. Setting focus position to the value read from HR_PID_CAM_FOCUS_POSITION_MAX will drive the focus motor to infinity focus. To command the lens to move to a new position close to the current position, the application should either add or subtract a small number of steps to the current position. The relationship of focus position steps and focus distance is not linear and will vary with different lenses. The number of steps allowed for focus adjustment will be scaled between the min and max values.

2.6.24 HR_PID_CAM_FOCUS_POSITION_MIN

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the value that will set the lens to the closest focus. This value will be zero for HR-1100c cameras.

2.6.25 HR_PID_CAM_FOCUS_POSITION_MAX

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the value that will set the lens focus to infinity. This value will be 10000 for HR-1100c cameras.

2.6.26 HR_PID_CAM_ISO

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Write	Camera

This controls the sensitivity of the camera in the same units as ISO film speed.

2.6.27 HR_PID_CAM_ISO_MIN

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the minimum value that can be set to HR_PID_CAM_ISO for this camera.

2.6.28 HR_PID_CAM_ISO_MAX

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This is the maximum value that can be set to HR_PID_CAM_ISO for this camera.

2.6.29 HR_PID_SHUTTER_SPEED_SECONDS

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Camera

This controls the amount of time the shutter is open, also known as “integration time”. The units are seconds, not 1 over seconds. For example, to get 1/60th second, the application should send the value 0.0166667.

2.6.30 HR_PID_SHUTTER_SPEED_SECONDS_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the minimum value that can be set to HR_PID_SHUTTER_SPEED_SECONDS.

2.6.31 HR_PID_SHUTTER_SPEED_SECONDS_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the maximum value that can be set to HR_PID_SHUTTER_SPEED_SECONDS.

2.6.32 HR_PID_CAM_APERTURE_F_NUMBER

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Camera

This controls and reports the aperture setting of the lens. Common examples are 2.8, 5.6, etc. Any float value within range is allowable. The lens aperture motor will be driven to the closest value.

2.6.33 HR_PID_CAM_APERTURE_F_NUMBER_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the minimum value that can be set to HR_PID_APERTURE_F_NUMBER.

2.6.34 HR_PID_CAM_APERTURE_F_NUMBER_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the maximum value that can be set to HR_PID_APERTURE_F_NUMBER.

2.6.35 HR_PID_CAM_TRIGGER_ENABLE

Data Type	Count	Access	Object
HR_TYPE_BOOL	1	Read-Write	Camera

This control parameter should be used by the application to:

- abort any pending trigger sequence,
- prevent an external hardware trigger from some external event, or
- disable triggering while other trigger control parameters are being changed (required)

If HR_PID_CAM_TRIGGER_ENABLE is true, the error code HR_ERR_CAM_TRIGGER_IS_ENABLED will be returned if the application calls HR_SetParam(...) for any of the following parameters:

- HR_PID_CAM_IMAGE_ACQ_MODE
- HR_PID_CAM_NUM_IMAGES_IN_SEQUENCE
- HR_PID_CAM_TRIGGER_DELAY_SECONDS
- HR_PID_CAM_TRIGGER_DISABLE_SECONDS
- HR_PID_CAM_STROBE_TRIGGERS_SELECTS
- HR_PID_CAM_STROBE_ADVANCE_SECONDS

The correct sequence for the Host Application is:

- HR_SetParam(camHandle, HR_PID_CAM_TRIGGER_ENABLE, &>false)
- HR_SetParam(...) for any or all of the above parameters
- HR_SetParam(camHandle, HR_PID_CAM_TRIGGER_ENABLE, &>true)

Conversely, if HR_PID_CAM_TRIGGER_ENABLE is false, the HR Library will return the error code HR_ERR_CAM_TRIGGER_IS_DISABLED if the Host Application calls

- HR_TriggerCamera(camHandle) or
- HR_SetParam(camHandle, HR_PID_CAM_RUN_PREVIEW, &>true)

The Host Application should not call HR_SetParam(camHandle, HR_PID_CAM_TRIGGER_ENABLE, &>false) while parameter HR_PID_CAM_RUN_PREVIEW is true (running). If so, the HR_SetParam() function will return HR_ERR_CAM_RUN_PREVIEW_IS_ON. No change will be made to the state of the HR Library or the camera.

If an image capture is triggered just before HR_PID_CAM_TRIGGER_ENABLE is set false, that image will not be stopped and will be transferred completely into the HR Library.

2.6.36 HR_PID_CAM_IMAGE_ACQ_MODE

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Write	Camera

Controls the camera trigger mode as specified by one of the enum values in HR_CamTriggerMode.

An attempt to change this value while HR_PID_CAM_TRIGGER_ENABLE is true will cause the HR_SetParam (...) function to return HR_ERR_CAM_TRIGGER_IS_ENABLED.

2.6.37 HR_PID_CAM_NUM_IMAGES_IN_SEQUENCE

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Write	Camera

With this value the application can control the number of images to be acquired for one sequence event. The minimum allowed value is 1, the maximum allowed value depends on the camera model and can be read from the HR Library using HR_PID_CAM_NUM_IMAGES_PER_SEQUENCE_MAX.

An attempt to change this value while HR_PID_CAM_TRIGGER_ENABLE is true will cause the HR_SetParam (...) function to return HR_ERR_CAM_TRIGGER_IS_ENABLED.

2.6.38 HR_PID_CAM_NUM_IMAGES_IN_SEQUENCE_MIN

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This value can be used by the application to determine the minimum number of images that can be acquired in a sequence event. The number can be different for different camera models. We expect this number to be 1.

2.6.39 HR_PID_CAM_NUM_IMAGES_IN_SEQUENCE_MAX

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This value can be used by the application to determine the maximum number of images that can be acquired in a sequence event. The number can be different for different camera models.

2.6.40 HR_PID_CAM_TRIGGER_DELAY_SECONDS

Data Type	Count	Access	Object
HR_TYPE_FLOAT	Variable	Read-Write	Camera

This array allows the application to control the time delay between a camera trigger event and the first image exposure (element zero) and the times between the start of successive image exposures in a sequence of images initiated by one trigger event if the application has set HR_PID_CAM_NUM_IMAGES_PER_TRIGGER greater than one. The size of this array will be HR_PID_CAM_MAX_NUM_IMAGES_PER_SEQUENCE and is based on the camera model. The HR Library will allow your application to set or read all the values in the array, even if your application does not use all of them because it does not change HR_PID_CAM_NUM_IMAGES_PER_SEQUENCE.

An attempt to change this value while HR_PID_CAM_TRIGGER_ENABLE is true will cause the HR_SetParam (...) function to return HR_ERR_CAM_TRIGGER_IS_ENABLED.

2.6.41 HR_PID_CAM_TRIGGER_DELAY_0_SECONDS_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the minimum value that should be set to the first element of the array of HR_PID_CAM_TRIGGER_DELAY_SECONDS. The trigger delay for the first image in a sequence can be significantly shorter than that for later images in a sequence.

2.6.42 HR_PID_CAM_TRIGGER_DELAY_X_SECONDS_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is one value that is the minimum value that should be set to any element, other than the first, of the array of HR_PID_CAM_TRIGGER_DELAY_SECONDS. The trigger delay for the first image in a sequence can be significantly shorter than that for later images in a sequence.

2.6.43 HR_PID_CAM_TRIGGER_DELAY_SECONDS_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the maximum value that should be set to any element of the array of HR_PID_CAM_TRIGGER_DELAY_SECONDS.

2.6.44 HR_PID_CAM_TRIGGER_DISABLES_SECONDS

Data Type	Count	Access	Object
HR_TYPE_FLOAT	Variable	Read-Write	Camera

This parameter allows the application to control the amount of time triggers are disabled after each exposure in a sequence in order to prevent a re-trigger from an external trigger source. The size of this array will be HR_PID_CAM_MAX_NUM_IMAGES_PER_SEQUENCE and is based on the camera model. The HR Library will allow your application to set or read all the values in the array, even if your application does not use all of them because it does not change HR_PID_CAM_NUM_IMAGES_PER_SEQUENCE.

An attempt to change this value while HR_PID_CAM_TRIGGER_ENABLE is true will cause the HR_SetParam (...) function to return HR_ERR_CAM_TRIGGER_IS_ENABLED.

2.6.45 HR_PID_CAM_TRIGGER_DISABLE_TIMED_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Camera

This parameter reports the minimum time value allowed for HR_PID_CAM_TRIGGER_DISABLE_SECONDS when HR_PID_CAM_IMAGE_ACQ_MODE is set to HR_ACQUISITION_TIMED.

2.6.46 HR_PID_CAM_TRIGGER_DISABLE_TIMED_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Camera

This parameter reports the maximum time value allowed for HR_PID_CAM_TRIGGER_DISABLE_SECONDS when HR_PID_CAM_IMAGE_ACQ_MODE is set to HR_ACQUISITION_TIMED.

2.6.47 HR_PID_CAM_TRIGGER_DISABLE_TRIG_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Camera

This parameter reports the minimum time value allowed for HR_PID_CAM_TRIGGER_DISABLE_SECONDS when HR_PID_CAM_IMAGE_ACQ_MODE is set to HR_ACQUISITION_TRIGGERED.

2.6.48 HR_PID_CAM_TRIGGER_DISABLE_TRIG_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Write	Camera

This parameter reports the maximum time value allowed for HR_PID_CAM_TRIGGER_DISABLE_SECONDS when HR_PID_CAM_IMAGE_ACQ_MODE is set to HR_ACQUISITION_TRIGGERED.

2.6.49 HR_PID_CAM_STROBE_TRIGGER_SELECT_MASK

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Camera

This value indicates which bits can be used as strobe triggers for HR_PID_CAM_STROBE_TRIGGERS_SELECTS. See related control below.

2.6.50 HR_PID_CAM_STROBE_TRIGGERS_SELECTS

Data Type	Count	Access	Object
HR_TYPE_INT32	Variable	Read-Write	Camera

This array allows the application to control which external strobe(s) the camera will fire for each exposure in a sequence, including a "sequence" with a count of 1. Each of the Strobe_Out pins on the J3 and J4 connectors on the back of the camera is driven by a specific bit of the parameter value(s). Which bits are available on a specific camera can be determined from GetParam(..HR_PID_CAM_STROBE_TRIGGER_SELECT_MASK)

The parameter control and hardware allow none, one, several, or all strobe select bits to be fired at any time. For example, the binary value 001001 (9 hex)will simultaneously fire strobe lines on pin 1 of both J3 and J4. The binary value 111111 (3f hex) will simultaneously fire all 6 strobe lines.

Binary	Hex	Connector	Pin
000001	0x01	J3	1
000010	0x02	J3	3
000100	0x04	J3	5
001000	0x08	J4	1
010000	0x10	J4	3
100000	0x20	J4	5

The size of this array will be HR_PID_CAM_NUM_IMAGES_IN_SEQUENCE_MAX which is based on the camera model. The HR Library will allow your application to set or read all the values in the array, even if your application does not use all of them because it does not change HR_PID_CAM_NUM_IMAGES_IN_SEQUENCE.

An attempt to change this value while HR_PID_CAM_TRIGGER_ENABLE is true will cause the HR_SetParam (...) function to return HR_ERR_CAM_TRIGGER_IS_ENABLED.

2.6.51 HR_PID_CAM_STROBE_ADVANCE_SECONDS

Data Type	Count	Access	Object
HR_TYPE_FLOAT	Variable	Read-Write	Camera

This array allows the application to control the time delay between the start of an exposure and the time the camera will fire an external strobe. The size of this array will be HR_PID_CAM_MAX_NUM_IMAGES_PER_SEQUENCE and is based on the camera model. The HR Library will allow your application to set or read all the values in the array, even if your application does not use all of them because it does not change HR_PID_CAM_NUM_IMAGES_PER_SEQUENCE.

An attempt to change this value while HR_PID_CAM_TRIGGER_ENABLE is true will cause the HR_SetParam (...) function to return HR_ERR_CAM_TRIGGER_IS_ENABLED.

2.6.52 HR_PID_CAM_STROBE_ADVANCE_SECONDS_MIN

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the minimum value that should be set to any element of the array of HR_PID_CAM_STROBE_ADVANCE_SECONDS.

2.6.53 HR_PID_CAM_STROBE_ADVANCE_SECONDS_MAX

Data Type	Count	Access	Object
HR_TYPE_FLOAT	1	Read-Only	Camera

This is the maximum value that should be set to any element of the array of HR_PID_CAM_STROBE_ADVANCE_SECONDS.

2.7 Parameter ID's for Image Objects

2.7.1 HR_PID_IMG_FILE_TYPE

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the HR_ImgDataFormatEnumType of the image.

2.7.2 HR_PID_IMG_WIDTH

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the width of the image in pixels.

2.7.3 HR_PID_IMG_HEIGHT

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the height of the image in scan lines.

2.7.4 HR_PID_IMG_CHANS_PER_PIXEL

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This value will be 3 for color images or 1 for monochrome images.

2.7.5 HR_PID_IMG_BYTES_PER_CHAN

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This value will be 1 or 2, depending on camera settings at the time the image was captured. For JPEG images this value will be 1.

2.7.6 HR_PID_IMG_WHITE_BALANCE

Data Type	Count	Access	Object
HR_TYPE_FLOAT	3	Read-Only	Image

This parameter reports the white balance triplet that was applied to the image data.

2.7.7 HR_PID_IMG_IMAGE_BUFFER_STRUCT

Data Type	Count	Access	Object
HR_BufferStructType	1	Read-Only	Image

This parameter reports the data buffer information about the image, specifically:

HR_PtrType dataPtr points to the beginning of the buffer
HR_Int32Type byteCount the byte count of the buffer allocation

Note: the allocation may be larger than the actual image data

2.7.8 HR_PID_IMG_WHOLE_IMAGE_BYTE_COUNT

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the data byte count of the complete image data. The value is the product of width * height * bytesPerChan * chansPerPixel.

This value is available as soon as the image transfer begins from the camera even though the complete image data is not yet present.

NOTE: If the image type is JPEG, the value from HR_PID_IMG_BYTES_COLLECTED will be much lower than the value from HR_PID_WHOLE_IMAGE_BYTE_COUNT.

2.7.9 HR_PID_IMG_COLLECTION_COMPLETE

Data Type	Count	Access	Object
HR_TYPE_BOOL	1	Read-Only	Image

This value will be false while the image is being transferred from the camera to the Image Object within the HR Library and will become true when the image transfer is complete.

2.7.10 HR_PID_IMG_BYTES_COLLECTED

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the number of bytes that have been transferred from the camera to the Image Object within the HR Library. This value will change as the image data is being transferred.

If HR_PID_IMG_COLLECTION_COMPLETE is true, HR_PID_IMG_BYTES_COLLECTED indicates the total byte count.

If the image type is JPEG, the value from HR_PID_IMG_BYTES_COLLECTED will be the byte count of the compressed data and therefore much lower than the value from HR_PID_WHOLE_IMAGE_BYTE_COUNT.

2.7.11 HR_PID_IMG_DATE_TIME_EXIF

Data Type	Count	Access	Object
HR_TYPE_STRING	20	Read-Only	Image

This parameter reports the date and time of image capture in EXIF format:
YYYY:MM:DD HH:MM:SS

The date-time string comes from the Operating System of the Host Computer. The EXIF format is useful because it is both human readable and can be sorted using strcmp() or STL string class compares. The count of 20 includes the terminating zero, consistent with the HR_TYPE_STRING, TIFF, and EXIF definitions.

2.7.12 HR_PID_IMG_SEQUENCE_NUMBER

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the sequence number of this image since the camera was last reset. The first image since camera reset will have a sequence number of zero. The sequence number of Preview images is separate from the sequence number for JPEG or TIFF images.

2.7.13 HR_PID_IMG_STRIP_COUNT

Data Type	Count	Access	Object
HR_TYPE_INT32	1	Read-Only	Image

This parameter reports the number of strips in the image data for non-JPEG images. Early versions of camera firmware and the HR Library will have the image data as one large strip, so this value will be 1. This value works in conjunction with HR_PID_IMG_STRIP_OFFSETS and HR_PID_IMG_STRIP_BYTE_COUNTS.

JPEG images will always have this value as 1.

2.7.14 HR_PID_IMG_STRIP_OFFSETS

Data Type	Count	Access	Object
HR_TYPE_INT32	variable	Read-Only	Image

This parameter reports the offset, in bytes, of each strip of image data from the start of the buffer. This is very similar to the "StripOffset" TIFF tag (tag 273) as defined in the TIFF standards. Early versions of camera firmware and the HR Library will have the image data as one large strip, so this array will have only one element, and the value in that one element is the offset to the first and only strip. This value works in conjunction with HR_PID_IMG_STRIP_COUNT and HR_PID_IMG_STRIP_BYTE_COUNTS.

JPEG images will always have this value as 0.

2.7.15 HR_PID_IMG_STRIP_BYTE_COUNTS

Data Type	Count	Access	Object
HR_TYPE_INT32	variable	Read-Only	Image

This parameter reports the byte count of each strip of image data. This is very similar to the "StripByteCount" TIFF tag (tag 279) as defined in the TIFF standards. Early versions of camera firmware and the HR Library will have the image data as one large strip, so this array will have only one element, and that one element will be the byte count of all the image data in the first and only strip. The application should not assume that the byte count of all strips are the same. This value works in conjunction with HR_PID_IMG_STRIP_COUNT and HR_PID_IMG_STRIP_OFFSETS.

JPEG images will always have this value as 0.

2.7.16 HR_PID_IMG_CAM_MODEL

Data Type	Count	Access	Object
HR_TYPE_STRING	variable	Read-Only	Image

This parameter reports the model string of the camera that acquired this image.

2.7.17 HR_PID_IMG_CAM_SERIAL_NUMBER

Data Type	Count	Access	Object
HR_TYPE_STRING	variable	Read-Only	Image

This parameter reports the serial number string of the camera that acquired this image.

2.7.18 HR_PID_IMG_CAM_NAME

Data Type	Count	Access	Object
HR_TYPE_STRING	variable	Read-Only	Image

This parameter reports the HR_PID_CAM_NAME string that was set to the camera at the time the camera acquired this image. The application controls the content of HR_PID_CAM_NAME.

2.7.19 HR_PID_IMG_CAM_HANDLE

Data Type	Count	Access	Object
HR_TYPE_CAM_HANDLE	1	Read-Only	Image

This parameter reports the handle number of the camera that acquired this image.

2.7.20 HR_PID_IMG_CAM_MAKE

Data Type	Count	Access	Object
HR_TYPE_STRING	variable	Read-Only	Image

This parameter reports the manufacturer of the camera that acquired this image.

2.7.21 HR_PID_IMG_CAM_FIRMWARE_VERSION

Data Type	Count	Access	Object
HR_TYPE_STRING	variable	Read-Only	Image

This parameter reports the firmware version of the camera that acquired this image.

Image File Formats

2.8 JPEG

When the HR Library is commanded by the Host Application to make JPEG files, the combination of an HR Camera and HR Library will create a JPEG formatted image in memory inside the DLL. The Host application needs only to write the data to a disk file to create a JPEG file. The data is in standard JPEG format, in the standard sRGB color space. This JPEG image file format is widely compatible with a large number of applications.

2.9 TIFF

To be documented.

Lens Control

An EF lens attached to the HR Camera can be controlled by the Host Application by sending commands through the SDK.

EF lenses with power focus can be commanded through the whole range of focus.

EF lenses with Aperture control can be commanded through the whole range of the lens aperture.

NOTE: Shutter control is not part of an EF lens. The HR Cameras use electronic shuttering.

2.10 Network Interface

Reference to the HR-1100c camera User Manual for camera installation and verification of operation over the interface.

2.11 Windows

The HR Demo Application has been designed and tested to run on the Microsoft ® Windows 2000 ® and Windows XP ® operating systems.

2.12 Linux

The HR Demo Application does not run on the Linux or Unix operating systems.

Revision History

Date	version	description	author
05-Oct-04	1.0.0	Started first draft	jrs
06-Oct-04	1.0.1	included Jon K's comments	jrs
12-Oct-04	1.0.2	added lots of details to the function reference section	jrs
12-Oct-04	1.0.3	included Kevin's comments	jrs
25-Oct-04	1.0.4	updated as per meeting 10/16/2004	jrs
27-Oct-04	1.0.5	added most function signatures but not all	jrs
16-Nov-04	1.0.6	fixed GetCameraList() and GetImageList() to use HR_Int32Type	jrs
23-Dec-04	1.0.7	added HR_GetErrorString(...)	jrs
29-Dec-04	1.0.8	included JK's and SS's comments	jrs
03-Jan-05	1.0.9	more detail in GetCameraList(...) and GetImageList(..)	jrs
01-Feb-05	1.0.10	more detail in HR_OpenCamera(...)	jrs
08-Feb-05	1.0.11	more detail in the trigger/timed parameters	jrs
09-Feb-05	1.0.12	added HR_Poll()	jrs
09-Feb-05	1.0.13	added comments about dynamic loading the DLL	jrs
03-Mar-05	1.0.14	put a page break before block diagram so it would not get split	jrs