Host Interface Specification

Digita™ Operating Environment
version 1.0

Part Number 6-2030-0040-01
Version 1.0
November 10, 1998

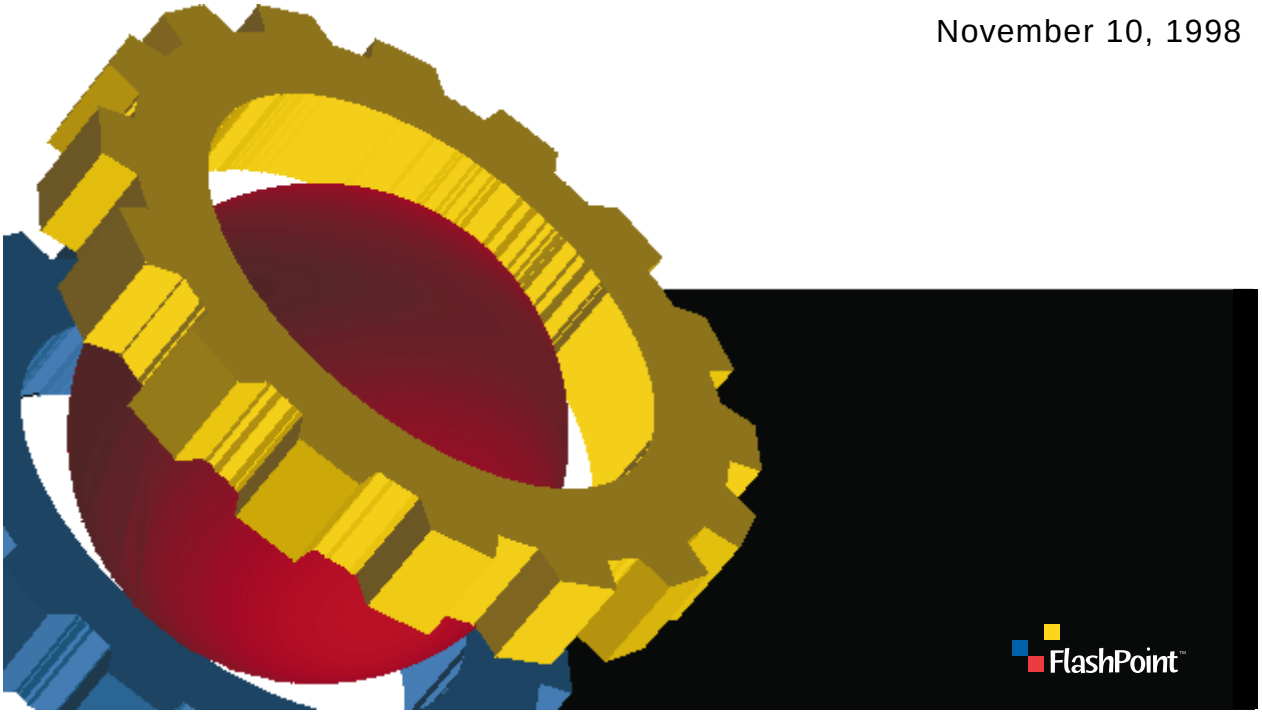


Table of Contents

1 Overview 1

Purpose and Audience 1

Camera Connect State 1

Hardware Protocol 1

Normal Speed Serial (NSS) 2

IrDA 2

Universal Serial Bus (USB) 2

Communication Flow Control 3

Beacon Phase 3

Baud Rate Changes 6

Flow Control 6

Packet Transport Mode 7

Camera Device Protocol Background 10

Camera Device Protocol 10

CDP Message Structure 10

Endian Problems 11

Using CDP 13

Image Access 15

2 Host Command Overview 17

Introduction 17

Core Command Group 17

General Core Commands 18

Product-Specific Core Commands 18

Product-Specific Command Group 18

Test Command Group 18

Reserved Command Group 18

Conventions 19

Format 19

Supported Primitive Types 19

Common Type Definitions 21
Common Error Codes 23
Parameter Ranges 24
Host Core Command Summary 24
Product and Image Information Commands 24
Status Commands 24
Camera Capabilities and State Commands 24
Scene Analysis Command 25
Power and Capture Commands 25
File Commands 25
Clock Commands 25
Serial Information Commands 26

3 Core Commands 27

Product and Image Information Commands 27
GetProductInfo 27
SetProductInfo 28
GetImageSpecifications 29
Status Commands 30
GetCameraStatus 30
GetError 33
Camera Capabilities and State Commands 34
GetCameraCapabilities 34
GetCameraState 35
SetCameraState 36
GetCameraDefault 36
SetCameraDefault 37
RestoreCameraState 38
Scene Analysis Command 38
GetSceneAnalysis 38
Power and Capture Commands 39
GetPowerMode 40
SetPowerMode 40
GetS1Mode 40
SetS1Mode 41
.StartCapture 41
File Commands 42
GetFileList 42

GetNewFileList 43
GetFileData 44
SetFileData 45
EraseFile 46
GetStorageStatus 46
GetFileTag 47
SetUserFileTag 47
Clock Commands 48
SetClock 48
GetClock 49
Serial Information Commands 49
GetInterfaceTimeout 49
SetInterfaceTimeout 50

Ch 1 Overview

This chapter provides an overview of the Digita™ operating environment host interface specification. The topics covered include the following:

- Purrpose and Audience
- Camera Connect State
- Hardware Protocol
- Communication Flow Control
- Image Access

Purpose and Audience

This manual provides the detailed host interface commands for the Digita operating environment, including data and control flow between the Digita device and the host. The intended audience is developers of software and firmware for Digita devices.

Camera Connect State

When the mode switch for a Digita device is set to “Connect,” the device enters that state. The Connect state is the only state in which a Digita device can communicate with a host. The display is off in the Connect state; other device characteristics for this state are device-specific.

Hardware Protocol

The following types of serial ports are available:

- NSS - Normal Speed Serial
- IrDA™ (Infrared Data Association™)
- USB (Universal Serial Bus)

Normal Speed Serial (NSS)

One serial interface for the camera to another device is an asynchronous NSS (Normal Speed Serial) port. The supported baud rates are 9600, 14400, 19200, 28800, 38400, 57600 and 115200 bps. These rates conform to the RS232C specification. The default data format (format of each byte) is 8-bit data with 1 stop bit and no parity bit. The least significant bit is sent first and the most significant bit is sent last. This data format cannot be changed. The serial communication is half duplex and asynchronous.

The NSS is a slave device.

IrDA

The Infrared Data Association (IrDA) serial port thread can be used for the following forms of communication:

- camera to printer
- camera to camera
- camera to computer

The current implementation is based on the IrDA lite standard with the following changes:

- the data transfer buffer is 2 kilobytes
- there is one slot discovery

For lower speed serial infrared (SIR), the data transfer rate can vary from 9600 baud to 115.2 kilobits per second. The protocol layers include the optional layers IrCOMM and TinyTP in addition to the required layers.

The IrDA can be either a master or slave device.

For more information on IrDA, refer to the web site http://www.irda.org.

Universal Serial Bus (USB)

The Universal Serial Bus can be used for communication between the camera and the host computer where there is a physical link between the two. Speed is negotiable, with a maximum of 12 megabits per second. The USB is a slave device.

For more information on USB, refer to the *Universal Serial Bus Specification*, Version 1.0, January 19, 1996.

Communication Flow Control

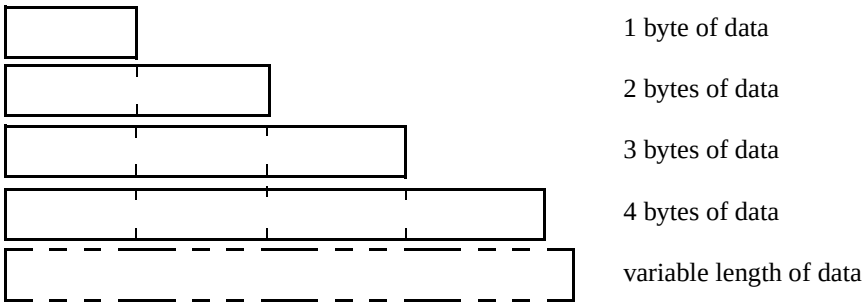
Note This section applies only to NSS connections.

The following sections describe how the communications works. The Beacon Phase is the initiation/negotiation phase used to determine the speed, protocol, and packet size. The Baud Rate Change section describes in more detail how to use a baud rate other than 9600. Flow Control describes how commands and responses are handled once the Beacon Phase is complete. Finally, Packet Transport Mode describes how the handshaking occurs after the Beacon Phase is done.

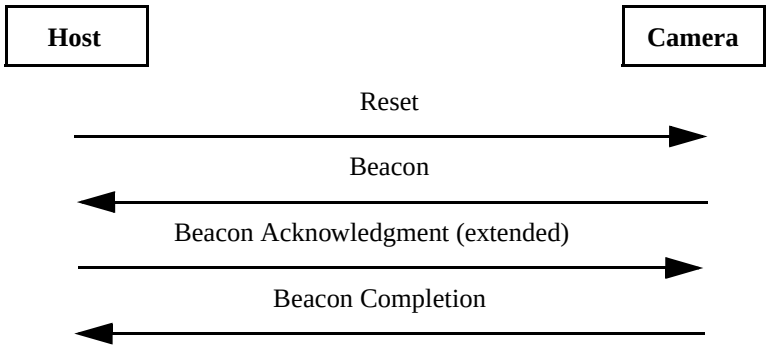
Beacon Phase

The beaconing phase is the initial communications between the host computer and the camera hardware. In this phase the host computer determines whether it wants to use NSS serial communication, and then the correct communications is set up based on the outcome. This beaconing phase uses 9600 baud. An overview of the Beacon Phase is provided below.

The following symbols are used in the diagrams in this section:



The following figures show the beaconing sequence:



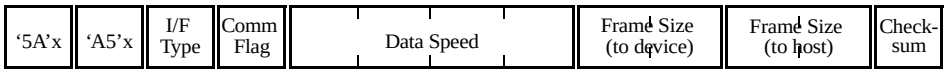
The Reset signal is a pulse on the Reset/Att line (which cooresponds to pin 2 at the camera side) sent from the host computer to the camera. This pulse must be at least 50us.

The beacon signal consists of the following bytes:

Where:

Vendor ID	‘xxxx’x - PCMCIA device id
Device ID	‘xxxx’x - controlled by the owner of the Device id; should be different for each product
Checksum	is calculated on all bytes, using bytewise modulo-2 addition.

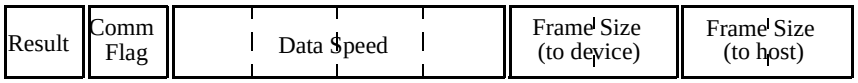
The beacon Acknowledgment signal consists of the following bytes:



Where:

I/F Type	‘55’x	
Comm Flag	bits 7-4	Reserved
	bits 3-2	Pod Receive mode
	bits 1-0	Host Receive mode
	‘00’x - Receiver supports polled transport mode with a length field	
Data Speed	The host computer should set the data speed to the maximum baud rate that it can support without errors.	
Frame Size (to device)	Maximum host to camera frame size	
Frame Size (to host)	Maximum camera to host frame size	
Checksum	Checksum is calculated on all bytes, using bytewise modulo-2 addition.	

The beacon Completion signal consists of the following bytes:

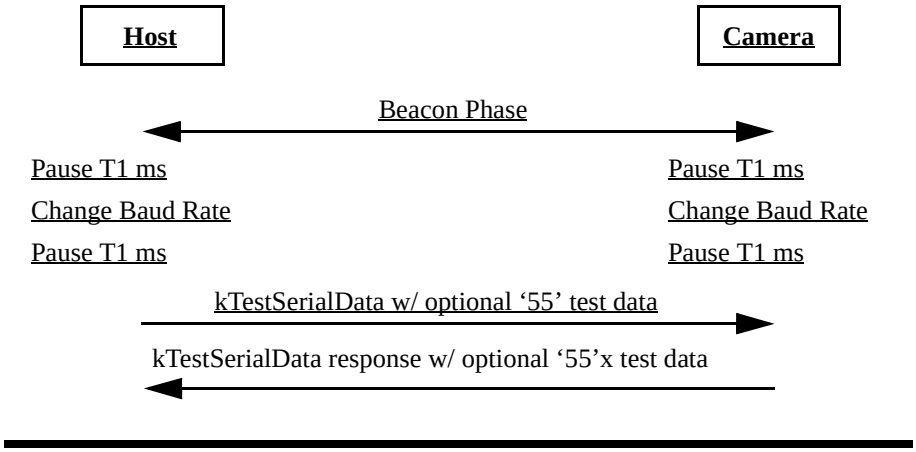


Where:

Result	0 - no errors	
	-1 - data rate incompatibility	
	-2 - device does not support the indicated receive modes	
	Result codes in the range of -128 to -64 are device-specific and may be assigned at a later time.	
Comm Flag	bits 7-4	Reserved
	bits 3-2	Pod Receive mode
	bits 1-0	Host Receive mode
Data Speed	The camera will set this to the fastest speed the camera can support while still being slower than the rate specified by the host in the Beacon Acknowledge.	
Frame Size (to device)	Maximum host to camera frame size	
Frame Size (to host)	Maximum camera to host frame size	

Baud Rate Changes

For asynchronous communications, the baud rate is changed after the beacon phase is completed. Both the Host computer and the camera switch to the agreed upon new baud rate after a specified time. The following figure shows the sequence of events needed to complete the sequence and, if desired, test that the baud rate works correctly.



If the camera detects errors in the data it receives from the `TestSerialData` command, the error code in the response back to the host will indicate this. The camera remains at this new baud rate until the host starts the beacon phase over again, at which time the camera will switch back to 9600 asynchronous communication. The host will then be responsible for specifying to the camera a slower rate and this cycle will start over again.

Note that the camera has an Idle timer so that if the host never responds after this sequence, the camera will switch back to 9600 asynchronous communication and wait for another beacon phase to start. Before the beacon phase starts, all data coming across the serial line is ignored. The host can set the value of the Idle timer, but only after completing the beacon phase.

The values of Pause T1 is 100 ms. The number of ‘55’x bytes to be sent to the camera from the host and to the host from the camera are specified in the `kTestSerialData` command.

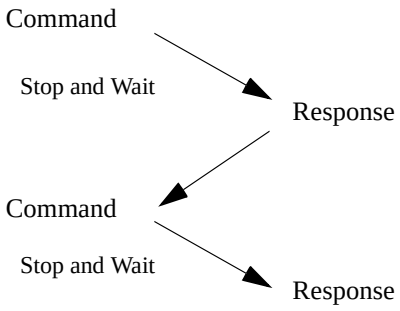
Flow Control

The flow control is of the stop-and-wait type. When a host sends a data block, it must “stop-and-wait” until it receives a response or acknowledgment before it is allowed to send the

next data block. If the command requires more than one packet to be sent, then it is transmitted in multiple packets. Another command can not be sent until a response is received from the camera. The response is also sent as a block in possibly multiple packets with the response bit in the flags bit set. The size of the packets is negotiated in the beacon phase.

The order of events in sending commands using stop-and-wait flow control is as follows:

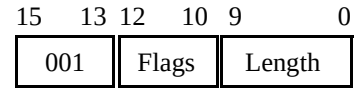
1. When the host has a Camera Device Protocol (CDP) command prepared and receives a response to a previous command or completes the beacon phase, it can send the command in one or more packets.
2. The camera then sends a CDP response when it has both prepared the response data and it is prepared to receive the next command. The response will be sent in one or more packets.
3. Once the host has received the response, it is free to send the next command as shown in the following diagram.



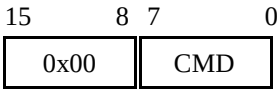
Packet Transport Mode

This mode is designed to support hosts incapable of receiving an unknown amount data at any given point in time. In this mode, the host first sends a poll with a length packet frame to get the device's attention. When the device is ready to receive data, the device replies with an acknowledge frame at which time the host sends the data frame.

The frame formats are shown below:



Poll With Length Packet Format



Acknowledge packet Format

Data Packet Format

The polled with length transport mode operates as follows: When side “a” wants to send data to side “b”, it sends a poll with length packet, indicating the size of the data packet it wants to send. When side “b” has received the poll with length packet and is ready to receive that size of data, it responds with an acknowledge packet. When side “a” receives the acknowledge packet it sends the data packet of the specified size plus 2 additional bytes for the follow on poll with length packet.

This mode assumes that the associated receiver is capable of receiving the two-byte poll and acknowledge packets at any time.

Maximum data packet size is negotiated during the beacon phase. Note that the maximum frame size chosen by designers is normally based on desired system response times and memory constraints.

The frame size is limited to 1023 bytes. This limits the time spent servicing the interface. The camera will reduce the frame size to 1023 if the host requests a larger frame size.

The time required for an endpoint to respond to an acknowledge is the acknowledge response time. This time is measured from the last bit of the closing flag of an acknowledge frame to the first bit of the first data byte in the data packet frame. A small acknowledge response time limits the amount of time the Host or Device must be attentive to the interface

after sending an acknowledge while it is waiting for data to arrive. The following time limits are suggestions and are not hard requirements inside the camera.

Table 1. Poll with Length Mode Ack Response Times

	Minimum	Maximum	Comment
Host	50 us	200 ms	The minimum acknowledge response time allows the receiving side time to set up to receive the data.
Device	50 us	20 ms	The minimum acknowledge response time allows the receiving side time to set up to receive the data.

The poll with length packet above is associated with a set of flags. The following flags are defined:

Table 2. Stream Flags

Bit Number		Description
Bit 0 (LSB)	BOB	A 1 in the Beginning Of Block bit indicates that the next packet is the first packet of a command or response which is always transferred as a block.
Bit 1	EOB	A 1 in the End Of Block bit indicates that the next packet is the last packet of a command or response.
Bit 2	Command/ Response	A 0 indicates that the next packet is a command and a 1 indicates that the next packet is a response.

The acknowledge packet above has a set of commands associated with it. The following commands are defined:

Table 3. Acknowledge Commands

Bit Number		Description
Bit 0 (LSB)	Ack	Indicates to the sender of a poll with length packet that the receiver of the poll with length packet is prepared to receive the associated data frame.
Bit 1	Nak	Indicates to the sender of a poll with length packet that the receiver does not want to receive the data associated with the poll with length packet. The sender should abort the send of the rest of the data, then reschedule and try again.

Camera Device Protocol Background

Note This section applies to all connections.

The Digita operating environment command set and Camera Device Protocols (CDP) let a host computer and camera communicate with one another remotely via a serial interface. This command set specifies both the operations to be performed and the format of the data transferred between the host computer and the camera. A serial interface is the transport for this command set—the user will choose a serial implementation from the set supported by the camera.

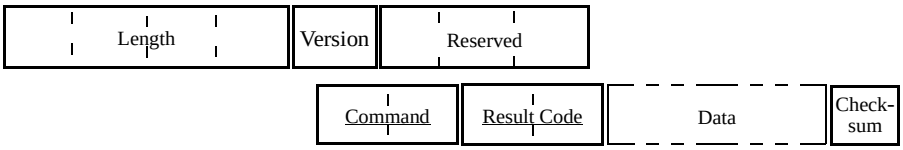
Camera Device Protocol

The Camera Device Protocol (CDP) is the protocol used to communicate between a Digita camera and the host computer. Several fields in this protocol allow for new features. The core structure of the CDP is based on a variable-sized message that contains a fixed-format header specifying a command length, version, command, result code, and optional data fields.

CDP Message Structure

The message structure of CDP is diagramed in Figure 1; a description of the structure follows.

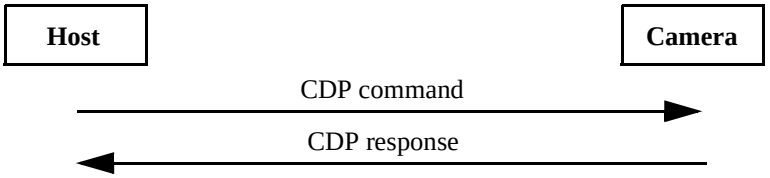
Figure 1. CDP Message Structure



Length	4 bytes	Indicates the total length of message not including the length field.
Version	1 byte	The version of this protocol.
Reserved	3 bytes	These are reserved for future enhancements.
Command	2 bytes	Indicates the command associated with the message
Result Code	2 bytes	Command result; this is ignored on request commands.
Data	varies	Message-specific data and parameters. This is required to start on a long word boundary.

The shortest command is 12 bytes, including 4 bytes for the length field, 1 version byte, 3 bytes reserved, 2 bytes for the command field, and 2 bytes for the result code field. The longest command would be over 4 Gbytes (plus 4 bytes for the length field).

All CDP communications between the host computer and a Digita camera use the following sequence of events.The header for CDP response echoes the protocol. The command field



contains the negative value (1’s complement) for the particular command to indicate that this is a response message. The result code indicates the status of the command. For example, if a camera doesn’t recognize a version or a command, the result code indicates it.

Endian Problems

One potential difficulty in supporting messages of variable length and data type is the interpretation of byte significance (least/most) when data is transferred between clients of differing byte order. Byte order in CDP messages is such that the most significant 8-bit byte in a short (16 bit) or long (32 bit) data element will occur earlier in the data stream than the remaining bytes of the short or long data type (this is called a Big-Endian representation). For example, the command field of the CDP header is defined as a int16 data type (16-bit, signed short), and the data stream representing the start of an CDP header would thus appear as follows:

<Most significant byte of Length><2nd Most significant byte of Length>< 3rd Most significant byte of Length><Least significant byte of Length><Version><Reserved bytes><High-order byte of command><Low-order byte of command><High-order byte of Result code><Low-order byte of Result code>

It is the responsibility of the host computer (software) to handle all byte-order swapping within an CDP message. This effectively means that on Windows® PCs some layer of software must be set up to perform conversion of the Big-Endian representation of a CDP message both to and from the Little-Endian byte order used most naturally on the host PC. The following collection of C-language macros handle the conversion between byte-ordering conventions. Keep in mind that in the non-Windows PC case these macros are essentially null operations.

Figure 2. Byte Order Conversion Macros

```
//=======================================================
//     Big-endian <-> little-endian Macro DEFINITIONS
//=======================================================
#define INT16   TShort
#define INT32   TLong

#if WINDOWS_PC
#define LOBYTE(w) (w & 0x00FF)
#define HIBYTE(w) ((w >> 8) & 0x00FF)
#define LB8(b)  b
#define BL8(b)  LB8(b)
#define LB16(w) (((INT16)(HIBYTE(w))) | ((LOBYTE(w) << 8) & 0xFF00))
#define BL16(w) LB16(w)
#define LB32(l) (((INT32)(LB16(HIWORD(l)))) | ((INT32)(LB16(LOWORD(l))) << 16))
#define BL32(l) LB32(l)
#else
#define LB8(b)  b
#define BL8(b)  b
#define LB16(w) w
#define BL16(w) w
#define LB32(l) l
#define BL32(l) l
#endif
```

Using CDP

The CDP command range is split into the four ranges given in Table 4.

Table 4. CDP Command Ranges

Use	Range	Size
Digita Operating Environment Core Host-to-Camera Commands	0x0000 - 0x5FFF	0x6000
Product Specific Host-to-Camera Commands	0x6000 - 0x6FFF	0x1000
Test Specific Host-to-Camera Commands	0x7000 - 0x7FFF	0x1000
Reserved	0x8000 - 0xFFFF	0x8000

The follow data structures are used for the CDP commands:

```
typedef struct {
     TULong       fLength;     // indicates the length of this
                               // structure minus the length of this
                               // field including the length of the
                               // data buffer associates with fData
     TUChar       fVersion;    // the version of the CDP packet
     TChar        fReserved[3]; // reserved
     TShort       fCommand;    // the command
     TShort       fResult;     // the result code
} TCOCDPHeader;

typedef struct {
     TCDPHeader   fCDPHeader;  // the header fields of the camera
                               // device protocol
     TUChar       fData[1];    // this is the start of the data to
                               // be sent with the message
} TCOCDP;
```

The following is the list of command codes for CDP operations that are supported within the Digita operating environment:

```
typedef enum {
     kCDPGetProductInfo         = 0x0001,
     kCDPGetImageSpecifications = 0x0002,
     kCDPGetCameraStatus        = 0x0003,
     kCDPSetProductInfo         = 0x0005,
     kCDPGetCameraCapabilities  = 0x0010,
     kCDPGetCameraState         = 0x0011,
     kCDPSetCameraState         = 0x0012,
     kCDPGetCameraDefaults      = 0x0013,
     kCDPSetCameraDefaults      = 0x0014,
     kCDPRestoreCameraStates    = 0x0015,
     kCDPGetSceneAnalysis       = 0x0018,
     kCDPGetPowerMode           = 0x0019,
     kCDPSetPowerMode           = 0x001a,
     kCDPGetS1Mode              = 0x001d,
     kCDPSetS1Mode              = 0x001e,
     kCDPStartCapture           = 0x0030,
     kCDPGetFileList            = 0x0040,
     kCDPGetNewFileList         = 0x0041,
     kCDPGetFileData            = 0x0042,
     kCDPEraseFile              = 0x0043,
     kCDPGetStorageStatus       = 0x0044,
     kCDPSetFileData            = 0x0047,
     kCDPGetFileTag             = 0x0048,
     kCDPSetUserFileTag         = 0x0049,
     kCDPGetClock               = 0x0070,
     kCDPSetClock               = 0x0071,
     kCDPGetError               = 0x0078,
     kCDPGetInterfaceTimeout    = 0x0090,
     kCDPSetInterfaceTimeout    = 0x0091,
} TCDPHostToCameraNewCommands;
```

The format used for describing the commands is as follows:

```
Command Name (ReqArgList) -> (ResArgList)
```

The `ReqArgList` defines what information is sent along with the command. The `ResArgList` is what the camera returns back to the host. In addition to this “data” response, any command can also return an error flag. Error flags will be returned for any command that is not in the correct format or a command requesting an action that is impossible or not understood.

Refer to “Host Core Command Summary” on page 24 for a summary list of the commands. The commands are described in detail in “Core Commands” starting on page 27.

Image Access

The file organization for image access is as follows:

```
Disk
     Imagesnn (first folder of images)
            FTIFOLD.INF (folder information file)
            IMnnnnnn.JPG (image file)
            TLnnnnnn (Time Lapse Natural Group folder)
                  IMnnnnnn.JPG (image file)
                  etc.
            HouseGrp (Programmed Group folder)
                  IMnnnnnn.JPG (image file)
                  BRnnnnnn (Burst Natural Group folder)
                        IMnnnnnn.JPG (image file)
            etc.
     Imagesnn (second folder of images)
     etc.
```

The top level has several folders that hold images. The folder is named *Imagesnn*, where *Images* is a name specified by a user preference (‘Images’ is the factory default), and *nn* is two digits, starting with 01.

There is a folder information file (FTIFOLD.INF) inside each ‘Images’ folder. This file specifies the serial number, make and model of the camera that took the images stored in that particular folder.

Within each camera folder are items, which can be either media files or groups. The groups can be either natural (e.g., time lapse, burst) or groups created by the user.

Still images are named IM*nnnnnn.TYP*, where *nnnnnn* is the image number for the camera and *TYP* is the file type (e.g., JPG). If there is a conflict with an existing file name when a new picture is captured, the existing file is renamed EX*nnnnnn.TYP*, where the *nnnnnn* is defined in the FTIFOLD.INF file and is incremented for each such occurrence of conflicting names.

Natural group folders have a two-letter designation followed by the image number of the first image in the group. For example, a time lapse that includes images IM000964 to IM001076 would be called TL000964. Here are the initial two-letter designations for such folders:

IM (varies)	Single image with or without sound (user-created group of still images). The two-letter designation is not necessarily “IM”; this is camera-specific.
BR	Burst
TL	Time lapse

The indicator *nnnnnn* is the image number for the camera. This number can be defined as follows:

- reset to 1 when the removable media is empty
- absolute camera count

Ch 2 Host Command Overview

This chapter provides an overview of the host command set. It describes the command access, command groups, command conventions, and provides a summary of available commands.

Introduction

The host command set is a set of commands that lets you access functions and features of cameras. This set of commands controls camera, system, and image processing operations, and is divided into four command groups—the *core group*, the *product-specific group*, the *test group*, and the *reserved group*. These groups are assigned specific command code ranges as summarized in Table 5.

Table 5. Command Types and Code Ranges

Command Type	Minimum	Maximum
Core Set	0x0000	0x5fff
Product-Specific Set	0x6000	0x6fff
Test Set	0x7000	0x7fff
Reserved	0x8000	0xffff

“Core Commands” starting on page 27 provides detailed information about the commands in this category.

Core Command Group

The first command group is the core command group. This group is the fundamental set of commands. The core group is divided into two sub-groups—general core commands, and product-specific core commands. Refer to “Core Commands” starting on page 27 for detailed information about the commands in this group.

General Core Commands

The general core command group is the common command set, and is supported on camera products.

Product-Specific Core Commands

The product-specific core commands are commands required of any camera system, but are dependent in some way on the hardware implementation. To use these commands from a host computer, the host software must request the proper vendor and product identification numbers from the command set, and execute the commands only if the identifiers are valid.

Product-Specific Command Group

The second group of commands is the product-specific command group. This group is a set of product-specific commands that are different based on the product manufacturer and product code. These commands are specified by the manufacturer for each product.

When using these commands in a host application, you must only call the commands in the specified code range for the camera product you are actually using. This command range allows camera vendors to differentiate their cameras from other cameras.

This publication does not cover this group of commands.

Test Command Group

The third command group is the test command group. These commands are reserved for camera testing during development, for manufacturer assembly line testing, and for testing when a camera is being serviced. Normally, these commands are not activated for use by a customer. However, a camera supplier can activate them for customer use as needed. In general, you should avoid using the test group code range within host applications since unpredictable results, including loss of data, may result.

This publication does not cover this group of commands.

Reserved Command Group

The final command group is the reserved command group. This is a range of commands reserved for internal command processing. You should avoid using the reserved group code range in your host applications.

Conventions

This section describes the command conventions for the core command set.

Format

The format used to describe commands in this document is as follows:

```
CommandName (ReqArgList) -> (ResArgList)
```

`ReqArgList`	Defines the information that is sent along with the command.
`ResArgList`	Specifies the data that the camera returns to the host. Each command also returns an error flag if the command is not in the correct format, or is requesting something that is impossible or not understood.

- All command values are transmitted in big endian format. For Windows®, host applications must re-order the bytes as needed to little endian format.
- All data is sent word-aligned. That is, 32-bit data is always aligned correctly, and 16-bit data is always aligned in a 32-bit data field.
- All `BitFlag` tables in this document list the bits in most significant bit order.
- Any additional data that is passed with required parameters will be ignored.

Supported Primitive Types

Table 6 shows the primitive types used in this document along with a description of the type.

Table 6. Supported Primitive Types

Type Name	Description
SInteger	32-bit signed integer
UInteger	32-bit unsigned integer
Fixed	32-bit signed fractional part in signed 15:16 format (15 bit signed integer and 16 bit fraction)
Boolean	32-bit Boolean, where non-zero is true and zero is false
BitFlags	32-bits of Boolean flags. Each bit can be either true or false.
PName	32-bit containing 4 ASCII characters

Table 6. Supported Primitive Types

Type Name	Description
DOSName	16-bytes containing a C string of up to 12 character in DOS® name format (including “.”)
ShortStr	16-bytes containing a string of up to 15 ASCII characters terminated by a null character.
String	32-bytes containing a string of up to 31 ASCII characters terminated by a null character.

Common Type Definitions

The following type definitions are common to all commands:

`ValueType` A `UInteger` value that specifies the type of data to follow. The possible values are shown below. The types supported by a particular command are listed in its description.

1, // Unsigned Integer (32 bits)

2, // Signed Integer (32 bits)

3, // Fixed (32 bits)

4, // Boolean (32 bits)

5, // BitFlags (32 bits)

6, // PName (32 bits)

7, // DOSName (16 bytes)

8, // String (32 bytes)

9, // Enumerated list of UInteger type with

// description strings

`ReqLength` A `UInteger` value that specifies the requested length sent to the camera from the host. Specifically, this value is the number of items in the list that is to follow. If the value is 0, no list follows.

`ResLength` A `UInteger` value that specifies the response length sent from the camera to the host as a response. Specifically, this value is the number of items in the list that is to follow. If the value is 0, no list follows.

`PName` A `PName` as defined in Table 6.
All camera standard `PNames` are lower case. All product-specific `PNames` must have at least one upper case character.

`PNameValue` Defines a data value for a `PName`. The `ValueType` specifies what kind of data is stored in the `PNameValue`. This data is one of the 8 `ValueTypes` as defined above.

`PNameValueStruct` `PName` and `PNameValue` pair. This is normally used in a `Set` command. The values must conform to the `ValueType` specified for each `PName` used. This is a structure defined as follows:

```
typedef struct {
    PName      Name;    // 4 character name
    PNameValue Data;    // the data
} PNameValueStruct
```

`PNameTypeValueStruct`

`PName`, `PNameType`, and `PNameValue` triplets. This is normally used in the response to a `Get` command.

```
typedef struct {
    PName               Name;// 4 character name
    PNameType           Type // one of 8 types
    union {
                UInteger UInt;//
                SInteger Int; //
                Fixed   fFixed;//
                Boolean  fBoolean;//
                BitFlags fBitFlags;//
                PName   fPName;//
                DOSName  fDOSName;//
                String   fString;//
    }
    PNameValue Data;        // the current setting
} PNameTypeValueStruct
```

`PNameTypeValueList`

Array of `PName TypeValueStructs`. This is normally used in the response to a `Get` command.

`FileNameStruct`

This is a Drive number, DOSName and path used to access any file in the file system.

```
typedef struct {
          Int       DriveNo; // drive number for
                             // this file
          String    PathName;// path for this file
                             // including closing
                             // '/' for example
                             // "CAMERA01/"
          DOSName   DOSName; // DOS name of file
} FileNameStruct
// DriveNo can be one of the following values:
typedef enum {
          kFSDriveA = 1,     // internal (RAM disk)
          kFSDriveB = 2,     // removable (IDE/ATA
                             // disk)
}
```

Note: The `PathName` and `DOSName` should be in uppercase to prevent possible disk corruption. However this will not be checked for.

Common Error Codes

This section describes the common error codes that may be passed back in the result field in response to a command. Table 7 lists the error code ranges.

Table 7. Command Error Code Ranges

Command Group	Minimum	Maximum
Core Set	0x0000	0x5fff
Product-Specific Set	0x6000	0xffff

The following common error codes are not repeated in each command's error code list:

`kCHNoErr (0)`	There was no error.
`kCHUnimplCommand (1)`	Illegal command; command not supported
`kCHUnsupportedVersion (2)`	Protocol error.
`kCHAppTimeout (3)`	Control Application failed to respond in time; timeout of interface.
`kCHInternalError (4)`	Memory errors, removable media read/write errors, bad file, image corrupted, operating system errors.
`kCHParamError (5)`	Illegal parameter value, too many parameters, too few parameters, bad format of parameter.
`kCHFileSystemFull (6)`	There is no more room on the file system specified.
`kCHFileNotFound (7)`	Specified file does not exist.
`kCHDataSectionNotFound(8)`	This image does not contain the section requested (thumbnail, audio,...).
`kCHInvalidFileType (9)`	The file specified is an invalid/unknown type.
`kCHUnknownDrive (10)`	The drive specified is unknown.
`kCHDriveNotMounted (11)`	The drive specified has nothing mounted.
`kCHSystemBusy (12)`	The system is currently busy and cannot currently process this command.
`kCHBatteryLow (13)`	Low battery condition.

Parameter Ranges

All parameter ranges shown in this document may vary from one camera product to another.To get the actual ranges for the commands for a given camera use the `GetCameraCapabilities` command.

Host Core Command Summary

The following is a summary of the host core command set. The commands are grouped by function. Each of these commands is described in detail in “Core Commands” starting on page 27.

Product and Image Information Commands

`GetProductInfo (PName) -> (ResLength, PNameTypeValueList)` page 27

`SetProductInfo (ReqLength, PNameTypeValueList) -> (DriveNo, Path)` page 28

`GetImageSpecifications () -> (ImageSpecificationList)` page 29

Status Commands

`GetCameraStatus () -> (SystemStatus, CaptureStatus, VendorStatus)` page 30

`GetError () -> (ErrorData)` page 33

Camera Capabilities and State Commands

`GetCameraCapabilities (PName) -> (ResLength, CapabilitiesData)` page 34

`GetCameraState (PName) -> (ResLength, PNameTypeValueList)` page 35

`SetCameraState (PNameValueStruct) -> ()` page 36

`GetCameraDefault (DefaultSource, PName) -> (ResLength, PNameTypeValueList)` page 36

`SetCameraDefault (UpdateSource, PNameValueStruct) -> ()` page 37

`RestoreCameraState (DefaultSource, PName) -> ()` page 38

Scene Analysis Command

GetSceneAnalysis (AnalysisType, Preview) -> (ResLength, SceneAnalysisData) page 38

Power and Capture Commands

GetPowerMode () -> (PowerState) page 40

SetPowerMode () -> () page 40

GetS1Mode () -> (s1Mode) page 40

SetS1Mode (s1Mode) -> () page 41

StartCapture () -> () page 41

File Commands

GetFileList (ListOrder, FileNameStruct) -> (ResLength, ResFileList) page 42

GetNewFileList () -> (ResLength, ResFileList) page 43

GetFileData (FileNameStruct, DataSelector, PartialTag) -> (PartialTag, FileData) page 44

SetFileData (FileNameStruct, DataSelector, PartialTag, FileData) -> () page 45

EraseFile (FileNameStruct) -> () page 46

GetStorageStatus () -> (TakenCount, AvailableCount, RawCount) page 46

GetFileTag (FileNameStruct, TagName) -> (ResLength, PNameTypeValueList) page 47

SetUserFileTag (FileNameStruct, PNameValueStruct) -> () page 47

Clock Commands

SetClock (ClockDateTime) -> () page 48

GetClock () -> (ClockDateTime) page 49

Serial Information Commands

GetInterfaceTimeout () -> (Timeout) page 49

SetInterfaceTimeout (Timeout) -> () page 50

Ch 3 Core Commands

This chapter describes the core commands. The commands fall into the following groups:

- “Product and Image Information Commands” beginning on page 27
- “Status Commands” beginning on page 30
- “Camera Capabilities and State Commands” beginning on page 34
- “Scene Analysis Command” beginning on page 38
- “Power and Capture Commands” beginning on page 39
- “File Commands” beginning on page 42
- “Clock Commands” beginning on page 48
- “Serial Information Commands” beginning on page 49

Each command is described in terms of syntax and parameter definitions. In cases where parameter descriptions apply only to one command, they are included in this chapter with the command description. However, many parameter descriptions apply to several commands. For a detailed list of camera parameters, refer to the appendixes of the *Script Reference*.

Product and Image Information Commands

The commands described in this section provide information about the camera and its imaging specifications.

GetProductInfo

This command requests specific information about the product by parameter. The information includes such things as the name of the camera vendor, the camera product name, hardware revision, and so forth. You can request information by parameter, or you can request all product information by sending a `PName` value of `NULL`, in which case, all `ProductInfo` parameters are returned.

The parameters are common to all cameras. You can add further `ProductInfo` parameters to the list for specific camera products. Any product-specific `ProductInfo` parameter

needs to have a least one upper case character. However, the host application can ignore product-specific `ProductInfo` parameters it does not understand.

Syntax

```
(PName) -> (ResLength, PNameTypeValueList)
```

Parameter Definitions

`PName` This specifies the `ProductInfo` item requested. If this is `NULL`, all `PNames` and their values will be returned.

`ResLength` This unsigned integer indicates the number of items sent from the camera to the host in `PNameTypeValueList`.

`PNameTypeValueList`
This list contains the `ProductInfo` items requested.

SetProductInfo

This command receives specific information about the connected product by parameter. The information includes such things as the name of the camera vendor, the camera product name, hardware revision, and so forth. The parameters that are required to be sent are: vdid, ptid, and sern. This camera will then create (if one doesn't already exist) a camera specific folder for images to be placed in. The path for this folder will be returned.

You can add further `ProductInfo` parameters to the list for specific camera products. The receiving camera can ignore any product specific parameters it does not know.

Syntax

```
(ReqLength, PNameTypeValueList) -> (DriveNo, Path, AvailableSpace)
```

Parameter Definitions

`ReqLength` This unsigned integer indicates the number of items received with this command in `PNameTypeValueList`.

`PNameTypeValueList`
This list contains the `ProductInfo` items received.

`DriveNo` This is the drive number for the following camera specific folder.

`Path` This string is the path for the camera specific folder.

`AvailableSpace` This signed integer is the size in bytes of the space available on the specified drive.

GetImageSpecifications

This command gets the hardware-related image information. This includes information about the camera's **zone** organization. Zone organization varies from camera to camera, so the information returned is product specific in terms of zone definition.

If type 0 is specified for zones, this implies a shorter `ImageSpecificationsList`, and no data is provided. Zones can be implemented as software or hardware, and that factor is not indicated in this section.

Syntax

```
() -> (ImageSpecificationList)
```

Parameter Definitions

`ImageSpecificationList` contains the following data, all of which are unsigned integer (`UInteger`) values.

CCD specifications

This parameter defines the following CCD features:
CCD pattern (1 = **Bayer** pattern, 2 and up are other patterns)
CCD pixels horizontal
CCD pixels vertical
CCD ring pixels horizontal
CCD ring pixels vertical
Number of bad columns,
Number of bad pixels.

Thumbnail specifications

This parameter provides the following thumbnail specifications:
Thumbnail Type (a value of 1 indicates YCC 422)
Thumbnail pixels horizontal
Thumbnail pixels vertical
Thumbnail file size

Screennail specifications

This parameter provides the following screennail specifications:
Screennail Type (a value of 0 indicates no screennail and a value of 1

indicates JPEG compressed)
Screennail pixels horizontal
Screennail pixels vertical

Focus zone specifications

This parameter provides the focus zone specifications:
Zone type (a value of 0 = none, 1 = rectilinear array, 2 and up indicate other types of arrays)
Number of horizontal zones
Number of vertical zones
Origin of horizontal zones
Origin of vertical zones
Horizontal zone size
Vertical zone size

Exposure zone specification

Zone type (a value of 0 = none, 1 = rectilinear array, 2 and up indicate other types of arrays)
Number of horizontal zones
Number of vertical zones
Origin of horizontal zones
Origin of vertical zones
Horizontal zone size
Vertical zone size

Status Commands

The commands described in this section provide information about camera status.

GetCameraStatus

This command returns three bit flags (`BitFlags`), `SystemStatus`, `CaptureStatus` and `VendorStatus`, that contain various status data from the camera. `GetCameraStatus` polls the camera on a regular basis, and the `BitFlags` indicate whether or not the internal state of the camera has changed. This means that all commands originate from the host, and if the camera needs service, it indicates this in the response to the command.

Syntax

```
() -> (SystemStatus, CaptureStatus, VendorStatus)
```

Parameter Definitions

The first bit flag, `SystemStatus`, contains up to 32 separate logical flags. If a camera cannot report some of these conditions, the related bit should be cleared and the condition reported as false. The bit positions do not change on this account, and all functions must be represented.The following flags are currently defined:

`ed1a`	Expansion disk 1 is installed and available. This is the most significant bit.
`ed2a`	Expansion disk 2 is installed and available.
`ramda`	RAM disk is installed and available.
`ipip`	Image Processing In Progress. If this bit is set, images are still being processed.
`memf`	When memory is full this flag is `true` (set to 1).. The flag is `false` when memory is not full.
`pwra`	Power alert indicates that the camera has entered a low-power state because of low batteries.
`flsc`	File list status change indicates that the file list from `GetFileList` has changed because of a successful capture. When this flag is set, you should call `GetNewFileList` or `GetFileList` to get a list of the changes. Either of these commands will clear this bit. This bit is set when processing of an image file has been sucessfully completed.
`trun`	Timer running indicates that the delay timer function is counting. `StartCapture` commands are not accepted at this time.
`stst`	Self test is in progress.
`stsc`	Self test is complete. This bit is cleared when the `SelfTest` command is issued to get the results from the self test.
`cerr`	Camera error indicates that an error has occurred in the camera. You should call `GetError` to find out what the error was and to clear this bit.
`mcro`	The macro flag indicates that the macro position is engaged. This can occur in response to a `SetCameraState` command or when you press the macro button on the camera.
`sbs1`	Shutter button in the S1 position indicates that you have pressed the shutter button half way down. This bit reflects the real-time status of the shutter button.

`sbs2`	Shutter button in the S2 position indicates that you have pressed the shutter button all the way down. This bit reflect the real-time status of the shutter button.
`zoom`	Zooming indicates that you are currently zooming the camera manually. This bit reflects the real-time status of the zoom buttons.
`extp`	External power indicates that external power is being applied to the camera.
`edch`	Expansion disk changed indicates that the expansion disk has been changed. This bit is cleared after a `GetFileList` command has been executed.
`rsvd`	The rest of these bits are reserved.

The second bit flag, `CaptureStatus`, contains up to 32 separate logical flags. The following flags are currently defined. If a camera cannot report some of these conditions, the related bit should be cleared and the condition reported as false. The bit positions do not change on this account, and all functions must be represented.

`camr`	Camera ready indicates that the camera subsystem is ready to capture a picture, and that any focus and exposure processing is complete. This is the most significant bit.
`strc`	Strobe charging indicates that the strobe subsystem is charging.
`strr`	Strobe ready indicates that the strobe is charged and ready to fire.
`shkw`	Shake warning indicates that the current (non-strobe) exposure requires a long exposure time and the camera should be held by a tripod or equivalent restraint.
`focl`	Focus locked is active when Focus Lock mode is selected and the shutter button is pressed into the S1 or S2 position. Under these conditions, the flag indicates when the automatic focus (AF) function has been completed.
`expl`	Exposure locked is active when Exposure Lock mode is selected and the shutter button is pressed into the S1 or S2 position. Under these conditions, the flag indicates when the automatic exposure (AE) function has been completed.
`oexp`	Overexposure indicates that with the current camera settings you will get overexposure.
`uexp`	Underexposure indicates that with the current camera settings you will get underexposure.

`stfa`	Subject too far away indicates that the AF system has determined the subject is too far away for the strobe to be effective.
`stcl`	Subject too close indicates that the AF system has determined that the subject is too close for the strobe to be effective or for the lens to focus on the subject properly. The focus system cannot distinguish between a subject that is too close or just on the edge of being too close, so if you focus on a subject at the minimum distance, this bit will be set.
`swbf`	Strobe will be fired. This is set at S1 if the strobe will be fired when S2 is pressed.
`chna`	Capture head not available.
`rsvd`	The rest of these bits are reserved.

The third bit flag, `VendorStatus`, contains up to 32 separate logical flags. These bit flags are defined by each individual vendor.

GetError

This command clears the error flag when the `GetCameraStatus` command returns a `cerr` flag. The command also returns the last error that occurred along with the description of the error.

Syntax

```
() -> (ErrorData)
```

Parameter Definition

`ErrorData` This parameter returns information about the error. The format is defined as follows:

```
typedef struct {
     UInteger     Date;
     UInteger     Time;
     SInteger     ErrorCode;
     String       ErrorDescription;
} ErrorData;
```

The hex encoding of the date and time is MM/DD/YY and HH/MM/SS. Thus, for 3/15/95 the value is 0x031595, and for 9:15:30 PM the value is 0x211530. Hours are in the 24-hour format.

The `ErrorDescription` string is `NULL`-terminated.

Camera Capabilities and State Commands

The commands described in this section provide information about

- the capabilities of the camera in terms of exposure mode, focus mode, and so forth
- camera defaults
- camera status

GetCameraCapabilities

This command requests the minimum and maximum values for any specified camera parameters, as well as the `ValueType` and factory default. The information is returned in `CapabilitiesData`. The command can request all available camera parameters by sending a `NULL` as the `PName`. For certain camera parameters, an enumerated list of strings is returned instead of a minimum and maximum value.

Syntax

```
(PName) -> (ResLength, CapabilitiesData)
```

Parameter Definitions

`PName`	This specifies the `CapabilitiesData` item requested. If this is `NULL`, all camera parameters and their capabilities will be returned.
`ResLength`	This unsigned integer indicates the number of items sent from the camera to the host in `CapabilitiesData`.
`CapabilitiesData`	This list contains the item requested in `PName` parameter or the entire list of `PNames`.

The following example of `CapabilitiesData` contains a list of `PNames`, along with their `ValueType`, and minimum, maximum, and factory default values. Each parameter has a text string that provides a descriptive name field for the parameter. Alternatively, for certain camera `PNames`, an enumerated list of values and text strings is sent (`ValueType = 9`), to allow for the correct camera mode to be selected by name.

A type of 1 through 5 implies a Range data type, a type of 6 - 8 implies a TypeValue data type, and a type of 9 implies a List data type.

```
typedef struct {
     PName        Name;         // 4 character name
     ValueType    Type;         // one of 9 types
     String       Descript;     // description of parameter
     union {
            ListData     List;         // if  specified as a list
            RangeData    Range;        // if specified as range
            ValueData    TypeValue;    // if specified type is a
                                       // value
     } Data;                           // start of the data
} CapabilitiesData
typedef struct {
     UInteger     Count;              // # of entries in the list
     UInteger     Factory;            // Factory Default;
     ListPair     Data[ ];            // array of size count
} ListData
typedef struct {
     UInteger     Value;        // value assigned to this description
     String       Descript;     // descriptive text
} ListPair
typedef struct {
     PNameValue   MinV;         // minimum value
     PNameValue   MaxV;         // maximum value
     PNameValue   Factory;      // factory default value
} RangeData
typedef struct {
     ValueType    Type;         // one of 9 types
     PNameValue   Factory;      // factory default for this type
} ValueData
```

GetCameraState

This command requests the current camera parameter settings by sending `PName`. Parameter settings are returned in `PNameTypeValueList`. If `PName` is `NULL` , all camera parameters are returned.

Syntax

```
(PName) -> (ResLength, PNameTypeValueList)
```

Parameters

`PName` This specifies the `PName` item requested. If this is `NULL`, all `PNames` and their current state will be returned.

`ResLength` This unsigned integer indicates the number of items sent from the camera to the host in `PNameTypeValueList`.

`PNameTypeValueList`
This list contains the item requested in `PName` parameter or the entire list of `PNames`.

SetCameraState

This command updates the current camera parameter settings. For range-type camera parameters, if a value set by the `SetCameraState` command is outside the allowable range specified by the `GetCameraCapabilities` command, an error is returned and no action is taken. If a value is set within the specified range, and the range is not continuous, the nearest value is set. For list-type camera parameters, if a value set by `SetCameraState` is not included in the list, an error is returned and no action is taken.

Syntax

```
(PNameValueStruct) -> ()
```

Parameter Definitions

`PNameValueStruct` This is a camera `PName` and the associated `PNameValue` pair. It indicates the value to be set. For all parameters that contain the word 'list,' for example `UIntegerList`, you should remove the word list, when the parameter is a `SetCameraState` parameter, so that the parameter is `UInteger`.

Camera parameters that are not relevant for specific exposure or focus modes are ignored, but saved for possible use when a different mode is selected. For example, a value for aperture can be set, but is ignored in Auto mode and used in Aperture Priority mode.

GetCameraDefault

This command requests the user or factory default values of the camera parameter settings by sending the `PName`. Parameter settings are returned in `PNameTypeValueList`. If `PName` is `NULL`, all camera parameters are returned.

Syntax

```
(DefaultSource, PName) -> (ResLength, PNameTypeValueList)
```

Parameter Definitions

`DefaultSource` This unsigned integer specifies which default to use. A value of 0 indicates that factory defaults should be used, while a value of 1 indicates that user defaults should be used.

`PName` This specifies the `PNameTypeValueList` item requested. If this is `NULL`, all `PNames` and their specified defaults will be returned.

`ResLength` This unsigned integer indicates the number of items sent from the camera to the host in `PNameTypeValueList`.

`PNameTypeValueList`
This list contains the items requested in `PName` parameter or the entire list of `PNames`.

SetCameraDefault

This command updates the user default values for the camera parameter settings. For range-type camera parameters, if a value set by the `SetCameraDefault` command is outside the allowable range specified by the `GetCameraCapabilities` command, an error is returned and no action is taken. If a value is set within the specified range, and the range is not continuous, the nearest value is set. For list-type camera parameters, if a value set by `SetCameraDefaults` is not included in the list, an error is returned and no action is taken. By setting `PName` parameter of the `PNameValueStruct` to `NULL`, all user defaults will be reset to either the current values or the factory defaults.

Syntax

```
(UpdateSource, PNameValueStruct) -> ()
```

Parameter Definitions

`UpdateSource` This unsigned integer specifies which source to use to update the user default value. A value of 0 indicates that the factory defaults should be used, a value of 1 indicates that the current values should be used, and a value of 2 indicates that the value in the `PNameValueStruct` should be used.

`PNameValueStruct` This specifies the parameter for which the defaults are to be set. If the `PName` parameter of this structure is set to `NULL`, all user defaults will be reset to the factory default or current values.

RestoreCameraState

This command restores either the user defaults or the factory defaults to the current value for all or a specified camera parameter. If PName is NULL, all camera parameters are restored to the specified defaults.

Syntax

```
(DefaultSource, PName) -> ()
```

Parameter Definitions

`DefaultSource` This unsigned integer specifies which source to use to update the current state value. A value of 0 indicates that factory defaults should be used, while a value of 1 indicates that user defaults should be used.

`PName` This specifies the parameter to be restored. If this is `NULL`, all parameters will be restored to either the user defaults or the factory defaults.

Scene Analysis Command

There is currently one scene analysis command, `GetSceneAnalysis`.

GetSceneAnalysis

This command returns information about the scene as analyzed by the camera. If the command is received prior to a complete analysis, the returned data is inaccurate. The host application, therefore, should check the ready status before issuing this command.

The camera can provide three types of information: camera settings, focus matrix, and exposure matrix.

Syntax

```
(AnalysisType, Preview) -> (ResLength, SceneAnalysisData)
```

Parameter Definitions

`AnalysisType` This unsigned integer `enum` value indicates which type of information is to be returned: `CameraSettings`, `FocusMatrix`, or `ExposureMatrix`.

1 = CaptureSettings

2 = FocusMatrix

3 = ExposureMatrix

`Preview` This Boolean value indicates whether the current viewfinder preview settings or the proposed normal capture settings are to be used. This parameter is ignored for `FocusMatrix`, or `ExposureMatrix`.

`ResLength` This unsigned integer indicates the number of items sent from the camera to the host in `SceneAnalysisData`.

`SceneAnalysisData`

This is an array of values that provides three types of information:

Camera Settings gives the camera settings of the camera proposed for a normal capture, unless `Preview` is true. In this case, the current preview (viewfinder) settings are returned. The setting values returned include EV (exposure valve), shutter speed, aperture, Flash Actuated, QuenchTime, QuenchLevel, 3 values for AWB (automatic white balance), AGC value, focus distance, zoom position, zoom speed, macro on/off, closest detected subject, and farthest detected subject. EV is reported as an `UInteger` value in 0.01 EV steps.

Focus Matrix and Exposure Matrix provide information about the array size of the Focus Matrix and Exposure Matrix, respectively. Array size is specified in the `GetSpecifications` command. The focus array returns focus distance per zone in centimeters. It returns 0 if no focus peak was detected. The exposure array returns EV values for each of the zones. EV values are specified in 0.01 EV steps. All matrix values are unsigned integers.

Power and Capture Commands

This section describes the commands associated with setting the camera's power mode, and initiating and finishing image capture.

GetPowerMode

This command determines the power level available to the camera.

Syntax

```
() -> (PowerState)
```

Parameter Definitions

`PowerState` This value contains unsigned integers from 1 to 5 that indicate the power state of the camera. Some devices will not have power state 4. The values contained are as follows:

5 = Full operation. Allows full operation of all camera functions.

4 = Reduced capture. This state applies in devices that have the option of low charge for flash. Image capture can occur with flash, but the time required to charge the flash is longer than in full operation mode.

3 = Minimal capture. This state disallows the use of the strobe entirely.

2 = Processing and Viewing only. does not allow the use of the image capture subsystem at all. You therefore cannot take any pictures.

1 = Processing only. This state only allow any image processing to be completed. The LCD is not on.

SetPowerMode

This command will power off the camera. When this command is issued from the host the camera will power down. There will be no response from the camera.

Syntax

```
() -> ()
```

GetS1Mode

This command will return the current state of the S1 button on the camera. This is taken from the current state in the CCS (not just a reading of the button position).

Syntax

```
() -> (s1Mode)
```

Parameter Definitions

`S1Mode`	This value indicates the current state of the S1 mode on the camera. 1 = S1 is on. 0 = S1 is off.

SetS1Mode

This command controls the overall operation of the camera by controlling the S1 position of the shutter button.

Syntax

```
(S1Mode) -> ()
```

Parameter Definitions

`S1Mode`	When this boolean is `true`, the camera will act as if the S1 button is depressed. When this boolean is `false`, the camera will act as if the S1 button is not depressed

.StartCapture

This command starts capture sequence. The camera will return the response right away. The host can determine when the image is ready to be downloaded and displayed by checking the Image Processing In Progress (`ipip`) flag in the `GetCameraStatus` command.

Syntax

```
() -> ()
```

Supplementary Information

Once a capture is complete, the `FileListStatusChange` flag is set for the `GetCameraStatus` command. The name of the file can then be retrieved with the `GetNewFileList` command. This clears the status bit.

File Commands

This section describes the commands associated with file management.

GetFileList

This command performs several functions.

- It allows the host to access the list of all captured files available in the current camera. For the default case, where the `FileNameStruct` specifies a `NULL DOSName`, this list includes any images that are stored in the camera. `ListOrder` specifies the returned list order: either ascending (order captured) or descending (reverse of order captured).
- In addition to the file list itself, this command also returns specific information about each file.
- This command can be used in response to the `FileListStatusChange` flag reported by the `GetCameraStatus` command. This flag is set when new files are created as the result of a `StartCapture` command. This command will clear this flag.

Syntax

```
(ListOrder, FileNameStruct) -> (ResLength, ResFileList)
```

Parameter Definitions

`ListOrder`	This unsigned integer specifies the order of the returned list: - A value 0 indicates that the return list specifies only 1 file - A value 1 indicates that the return list is in ascending order; a value of 1 is illegal if `FileNameStruct` specifies a particular file - A value 2 indicates that the return list is in descending order; a value of 2 is illegal if `FileNameStruct` specifies a particular file
`FileNameStruct`	This is the `DOSName` and path for the file for which information is requested. If a name is not recognized, an error is returned, and no `ResFileItem` is included in the `ResFileList`. If the `DOSName` is `NULL`, all images stored in the camera will be returned.If the filename

is VIEWFIND, then the last thumbnail size image that was taken will be returned.

ResLength — This unsigned integer indicates the number of items in ResFileList.

ResFileList — This is a hierarchical list of images in the camera. Information about each image contains the DOSName along with BitFlags defining the state of the file. The following structure shows a ResFileList.

```
Typedef struct {
       SInteger     DriveNo;     // drive number
       String       PathName;    // path for this
                                 // image including ‘/’
       DOSName      DOSName;     // DOS name of file
       SInteger     FileLength;  // length of file in
                                 // bytes - 1 if unknown
       BitFlags     FileStatus;  // File status BitFlags
} ResFileItem
```

Table 8 defines the FileStatus BitFlags. Any unused bits are reserved.

Table 8. FileStatus BitFlags

Name	Description
ipcm	Image processing complete. A value 1 is true; a value 0 is false

GetNewFileList

This command is used in response to the FileListStatusChange (flsc) flag reported by the GetCameraStatus command. This flag is set when new files are created from the result of a StartCapture command. This command returns the new filenames and clears the flag.

Syntax

```
() -> (ResLength, ResFileList)
```

Parameter Definitions

ResLength — This unsigned integer indicates the number of items in ResFileList.

ResFileList — This is a list of ResFileItems containing the DOSName for each file, along with BitFlags defining the state of the file.

GetFileData

This command is used to retrieve a thumbnail, or an entire capture file. Depending on the file type, a file can contain not only compressed image data, but also a thumbnail.

When retrieving an entire capture file, because of limited buffer space in the device we are connected to, the data set is too large to transfer with a single `GetFileData` command. The maximum size of the data set that can be returned is specified by the `mhbs` (Maximum Host Buffer Size) camera parameter. To overcome this limitation, the `GetFileData` command performs entire file transfers as a series of partial data sets.

Note The `GetCameraStatus` command returns a bit indicating when the camera memory is full.

Syntax

```
(FileNameStruct, DataSelector, PartialTag) -> (PartialTag, FileData)
```

Parameter Definitions

`FileNameStruct` The `DOSName` and path of the file received by means of the `GetFileList` command.

If the `DOSName` is "`VIEWFIND`", the previously captured thumbnail will be returned. This works in conjunction with `StartCapture` with the `CaptureType` set to `Thumbnail`.

`DataSelector` This unsigned integer allows the host to select either the entire file, or the thumbnail. On the first transfer `Offset` should be set to 0. The camera returns its next request values in the `PartialTag` structure. If the returned `Offset` is 0, the next set of data is from the beginning of the file.

While partials are requested in order, the actual position of the data within the file is specified in the response as part of `PartialTag`. If an error occurs during transfer, the partial may be re-requested.

Note that a thumnbail or screennail image must be retrieved with a single `GetFileData` command. Thus, the host buffer must be set larger than the thumbnail or screennail image size, which typically may be 15000 bytes. The default setting for the buffer is 19200 bytes.

`DataSelector` selects the data requested in the following ways:

0 = Get entire file; this is one data type that requires partial transfers

1 = Get thumbnail only

`PartialTag` This tag identifies the portion of the image returned. The tag includes the (byte) offset within the data set, and a length. The `Offset` can be used to re-request a partial data set from the camera once it has been received by the host, assuming an error has occurred.

```
typedef struct {
       UInteger     Offset;      // Defines starting
                                 // position of data
                                 // within file
       UInteger     Length;      // Defines byte
                                 // count of data
       UInteger     Filesize;    // the total size
                                 // of the file
} PartialTag
```

`FileData` This is the actual data requested. For the thumbnail image the following data structure is returned:

```
typedef struct {
       UInteger     dataSize;    // length of data
                                 // returned
       UInteger     height;      // height in pixels
       UInteger     width;       // width in pixels
       UInteger     type;        // format of the data
                                 // returned
       char         data[];      // beginning of the
                                 // data
} ThumbnailData
```

SetFileData

This command is used to put a file in the camera's file system.

When writing a file, because of limited buffer space in the device we are connected to, the data set could be too large to transfer with a single `SetFileData` command. The maximum size of the data set that can be written is specified by the `mhbs` (Maximum Host Buffer Size) camera parameter. To overcome this limitation, the `SetFileData` command performs entire file transfers as a series of partial data sets.

Syntax

```
(FileNameStruct, DataSelector, PartialTag, FileData) -> ()
```

Parameter Definitions

`FileNameStruct`	This `DOSName` and the directory path.
`DataSelector`	This unsigned integer allows the host to select the entire file. While partials are requested in order, the actual position of the data within the file is specified in the command as part of `PartialTag`. `DataSelector` selects the data requested in the following ways: 0 = Set entire file; this is one data type that may require partial transfers
`PartialTag`	This tag identifies the portion of the file written. The tag includes the (byte) offset within the data set, and a length.
`FileData`	This is the actual data sent.

EraseFile

This command allows the host to delete image files .

Syntax

```
(FileNameStruct) -> ()
```

Parameter Definition

`FileNameStruct`	This `DOSName` and path received by means of the `GetFileList` command.

GetStorageStatus

This command determines the status of storage available to the camera. The command always returns two `UInteger` values: `TakenCount`, and `AvailableCount`, and one `SInteger RawCount`

Syntax

```
() -> (TakenCount, AvailableCount, RawCount)
```

Parameter Definitions

`TakenCount`	This unsigned integer indicates how many capture files exist.
`AvailableCount`	This unsigned integer indicates an approximate number of compressed files can fit in the free storage.
`RawCount`	This signed integer indicates how many unprocessed files can fit in the available space. If the camera does not store the raw image data this value will be set to -1.

GetFileTag

This command returns the value of the specified file tag. If the `PName` is `NULL`, the entire list of file tags is returned.

Syntax

```
(FileNameStruct, PName) -> (ResLength, PNameTypeValueList)
```

Parameter Definitions

`FileNameStruct`	This `DOSName` and path of the file received by means of the `GetFileList` or `GetNewFileList` command.
`PName`	Specifies which file tag is requested. If this is `NULL`, the entire list of file tags is returned. Refer to the appendixes of the *Script Reference* for the parameters currently available.
`PNameValueList`	This is an array of `PNameValueStruct`. The list contains the item requested in `PName` parameter or the entire list of `PNames`.

SetUserFileTag

This command sets the value of the specified user tag. This applies to the tags that start with ‘u’.

Syntax

```
(FileNameStruct, PNameValueStruct) -> ()
```

Parameter Definitions

`FileNameStruct` This `DOSName` and path of the file received by means of the `GetFileList` command.

`PNameValueStruct` This contains the item the user wants to set and the value.

Clock Commands

This section describes the clock commands that allow you to set and get the current time and date.

SetClock

This command sets the current clock value from `ClockDateTime` as MM/DD/YY (month/day/year) and HH/MM/SS (hour/minute/second). `SetClock` does not "filter" dates in an attempt to verify whether the date/time parameters sent from the host are valid. It is the responsibility of the host program to intercept any invalid dates or times which the user might attempt to enter (such as trying to set the date to Feb. 31). If an invalid date or time is sent to the camera the result is undefined.

Syntax

```
(ClockDateTime) -> ()
```

Parameter Definition

`ClockDateTime` This structure, as shown in the following example, contains two unsigned integer values that have the hex encoding of MM/DD/YY and HH/MM/SS. For example, for 3/15/95 the value is 0x031595, and for 9:15:30 PM the value is 0x211530. Note that hours are in the 24-hour format.

```
typedef struct {
      UInteger     ClockDate;
      UInteger     ClockTime;
} ClockDateTime;
```

GetClock

This command gets the current date and time. All zeros are returned in `ClockDateTime` if the clock is not set or not running. Otherwise, the command gets the current state of the clock.

Syntax

```
() -> (ClockDateTime)
```

Parameter Definition

`ClockDateTime` This structure contains two unsigned integer values that have the hex encoding of MM/DD/YY and HH/MM/SS. For example, for 3/15/95 the value is 0x031595, and for 9:15:30 PM the value is 0x211530. Note that hours are in the 24-hour format.

Serial Information Commands

This section describes the commands that relate to control of the serial port and serial data.

GetInterfaceTimeout

This command returns the current settings in the `Timeout` structure as set by the `SetInterfaceTimeout` command.

Syntax

```
() -> (Timeout)
```

Parameter Definitions

`Timeout` This structure contains the basetime for `HostPollTimeOut` and `CmdRspTimeOut` which are measured in seconds. The format is defined as follows:

```
typedef struct {
       UInteger     HostPollTimeOut;
       UInteger     CmdRspTimeOut;
} Timeout;
```

`HostPollTimeOut` is the maximum amount of time in seconds that may elapse between two consecutive commands from the host. (The default value for `HostPollTimeOut` is 10.) If this time is exceeded the serial interface determines that the host is no longer connected. This forces the host to start the connection process again for any additional commands. It also allows the camera to take over the timing delay before powering down the camera.

`CmdRspTimeOut` is the amount of time in seconds that the serial interface thread in the camera will wait for the control application to respond to a command. (The default value for `CmdRspTimeOut` is 5.) If this time is exceeded, the serial interface thread sets the result code to indicate that a response timeout had occurred and the camera then resets itself. This forces the host to start the connection process again.

SetInterfaceTimeout

This command allows the host to set the various serial port `TimeOut` values.

Syntax

```
(Timeout) -> ()
```

Parameter Definitions

`Timeout`	This structure contains the basetime for `HostPollTimeOut` and `CmdRspTimeOut`, measured in seconds. If these values are set to 0, the camera will not change from its current value. The format is defined as in `GetSerialInfo`. (The current default value for `HostPollTimeOut` is 10; for `CmdRspTimeOut`, 5.)