SBIG
ASTRONOMICAL
INSTRUMENTS

SANTA BARBARA INSTRUMENT GROUP
P.O. Box 50437 • 1482 East Valley Road, Suite #33
Santa Barbara, CA 93150
Phone (805) 969-1851 FAX (805) 969-4069
e-mail:sbig@sbig.com home page:www.sbig.com

Application Note
STV Command Code Protocol
5th Draft Version
January 20, 2000

1. Introduction

This document describes the software interface to the STV camera. This camera consists of two components: an Optical Head and a CPU. The CPU interfaces to the Host computer through three-wire RS-232. The CPU is DSP based with internal Flash based firmware that controls the operation of the Optical Head and provides frame storage capability for captured images. Additionally the CPU contains a keypad, an electroluminescent character display and an optional LCD Video Monitor.

The CPU communicates to the Host computer through a packet-oriented protocol. The main purpose of these communications is so the Host computer can act like a "Remote Terminal", echoing the contents of the CPU's character display and reporting remote keyboard events.

2. Packet Structure

The data sent from the STV to and from the Host are structured into packets. The packets contain additional information to aid in the receipt and detection of transmission errors. The packet structure is shown below:

A5H = Start Byte
XXH = Command Byte
XXH = Data Length N (Low Byte)
XXH = Data Length N (High Byte)
XXH = Header Checksum (Low Byte)
XXH = Header Checksum (High Byte)
XXH = Data Byte 1
XXH = Data Byte 2
.
.
.
XXH = Data Byte N
XXH = Data Checksum (Low Byte)
XXH = Data Checksum (High Byte)

Several items about the packet structure are worth noting:

- All packets start with the Start Byte A5 hexadecimal.
- There are 256 possible commands due to the one-byte command.
- The length bytes contained in the packet tell the data length. If there is no data the packet length is 6 bytes. If there are data the packet length is 8 plus the Data Length.
- There are one or two Checksums. The Header Checksum is always present and is the 16 bit unsigned sum of the first 4 bytes in the data packet. The Data Checksum is

only present if data follows the header and is the sum of the data bytes. The checksums are modulo 65536, meaning if the sum overflows 16 bits it just wraps around.

In addition, 2 header-only packets are defined to signify Acknowledgement (ACK) and Negative Acknowledgement (NACK) of data:

Command 06H = ACK (Valid packet received)
Command 15H = NAK (Invalid packet received)

3. Serial Data Format

As shipped from the factory the CPU expects communications with the Host computer over its COM port at 9600 baud. The baud rate can be reprogrammed for other data rates and the settings are retained in nonvolatile memory. The data format is 8 data bits, no parity bits, and 1 stop bit (a total of 10 bits per byte including the start bit).

The following sections describes the individual commands that the Host computer can send to or receive from the CPU. Each command has a name and a command byte (the command byte is the 2nd byte in the packet). Individual commands can have zero or more data bytes (depending on the command itself). If the command has no data then the Data Length bytes in the packet are both zero. Data after the header is described in terms of the following parameter types:

byte	Unsigned 8 bit integer.
BOOL	Two bytes in length, least significant byte first. A value of 1 designates TRUE and a value of 0 designates FALSE.
BUFFER	Enumerated unsigned integer, two bytes in length, least significant byte first. Values 0 thru 29 designate possible FLASH buffers 1 thru 30 and values 30 and 31 represent the current DARK, LIGHT buffers.
enum	Enumerated unsigned integer value with a set of allowed values. Two bytes in length, least significant byte first.
UINT	Unsigned integer, two bytes in length, least significant byte first.
signed int	Same as int only signed.
ULONG	Unsigned long integer, 4 bytes in length, least significant byte first followed by the next 3 bytes in order of increasing significance.
signed long	Same as long only signed.

Some commands sent from the Host to the STV elicit only an ACK response from the where as other commands involve the CPU data back to the Host computer

Each of the CPU's commands is described in the sections below. The commands have been broken down into logical groups.

5. STV Generated Data

Comm Events

Request Ack - Command 0x10, Contains data

The Request Ack command is sent by the host to the STV to establish a communications link. The STV responds with an ACK.

Display Events

Display Echo - Command 0x09, Contains data

This command can be used to have the STV report the contents of the Alphanumeric Display. The command takes a single BOOL that when TRUE causes the contents of the display to be reported upon receipt of the command and each time the display changes. The response and subsequent broadcasts contain 48 bytes; the two rows of 24 characters on the display. Sending the Display Echo command with the data FALSE causes the STV to stop reporting the display contents and the STV will respond with an ACK.

Track Report

File Ops Events

STV Request Download - Command 0x00, No Data

The STV Request Download Event is sent by the STV when the user, sitting at the STV (remote from the host) wants to Download an image from the STV to the remote host. If the host is unready for data it should respond with the same command back to the STV (Command 0x01 with no data). Otherwise the host should respond with a request for image data within 1 second to indicate it will start the download process, which would involve downloading the image info and image from the LIGHT buffer.

STV Request Download All - Command 0x01, No Data

The STV Request Download All Event is sent by the STV when the user, sitting at the STV (remote from the host) wants to Download all the images from the STV to the remote host. If the host is unready for data it should respond with the same command back to the STV (Command 0x01 with no data). Otherwise the host should respond with a request for image data within 1 second to indicate it will start the download process which involves requesting the flash buffer status then downloading those buffers that contain data.

File Status - Command 0x0B, Contains Data

The File Status command is sent by the Host to the STV to indicate to the STV that status of the Hosts' file system in response to the STV Request Download All command. The command has one word of data with 0 indicating the File System is Ready (OK) and 1 indicating Not Ready. If the STV receives a failure notice it will display a message saying the remote server is not ready to accept data and halt the download. This command is sent by the Host after each image has been downloaded to indicate the data was successfully saved on disk.

Download Complete - Command 0x02, No Data

The STV Download Complete command is sent by the Host to the STV to indicate the STV initiated download (STV Request Download or STV Request Download All) is complete. The STV will respond with an ACK.

Request Buffer Status - Command 0x03, No Data

The Request Buffer Status is sent by the host to the STV to get the status of the Flash and working image buffers. The response by the STV is the following set of bits where 1 indicates that the buffer contains data and 0 indicates the buffer is empty:

```
UINT status1;       /* set of bits, b15=LIGHT, b14=DARK,b13-b0=Flash 30-17 */
UINT status2;       /* set of bits, b15-b0=Flash 16-0 */
```

Request Image Info - Command 0x04, Contains Data

The Request Image Info is sent by the host to the STV to get the Image Info associated with the image being downloaded. The request has the following data:

```
BUFFER buffer;      /* buffer to query, 0-29 = Flash, 30=Dark, 31=Light */
```

The STV replies with the same command with the following data format:

```
/*
      descriptor fields - these are sets of bits saved in
      the descriptor field of the IMAGE_INFO struct

*/
#define ID_BITS_MASK          0x0001      /* mask for no bits */
#define ID_BITS_10            0x0001      /* image is full 10 bits */
#define ID_BITS_8             0x0000      /* image from focus, 0nly 8 bits */
#define ID_UNITS_MASK         0x0002      /* mask for units of scope */
#define ID_UNITS_INCHES       0x0002      /* units were inches */
#define ID_UNITS_CM           0x0000      /* units were cm */
#define ID_SCOPE_MASK         0x0004      /* mask for telescope type */
#define ID_SCOPE_REFRACTOR    0x0004      /* scope was refractor */
#define ID_SCOPE_REFLECTOR    0x0000      /* scope was reflector */
#define ID_DATETIME_MASK      0x0008      /* mask for date/time valid */
#define ID_DATETIME_VALID     0x0008      /* date/time was set */
#define ID_DATETIME_INVALID   0x0000      /* date/time was not set */
#define ID_BIN_MASK           0x0030      /* mask for binning mode */
#define ID_BIN_1X1            0x0010      /* binning was 1x1 */
#define ID_BIN_2X2            0x0020      /* binning was 2x2 */
#define ID_BIN_3X3            0x0030      /* binning was 3x3 */
#define ID_PM_MASK            0x0400      /* mask for am/pm in time */
#define ID_PM_PM              0x0400      /* time was pm, add 12 hours */
#define ID_PM_AM              0x0000      /* time was am, don't add 12 hrs */
#define ID_FILTER_MASK        0x0800      /* mask for filter status */
#define ID_FILTER_LUNAR       0x0800      /* lunar filter was used for image */
#define ID_FILTER_NO          0x0000      /* no filter was used for image  */
#define ID_DARKSUB_MASK       0x1000      /* mask for dark subtraction */
#define ID_DARKSUB_YES        0x1000      /* image was dark subtracted */
#define ID_DARKSUB_NO         0x0000      /* image was not dark subtracted */
#define ID_MOSAIC_MASK        0x6000      /* mask for mosaic status */
#define ID_MOSAIC_NONE        0x0000      /* no mosaic, one image per frame */
#define ID_MOSAIC_SMALL       0x2000      /* small mosaic:40x40 pixels/image */
#define ID_MOSAIC_LARGE       0x4000      /* large mosaic:106x100 pixels/image */

/*

      IMAGE_INFO - this records data for the current image

      Notes:
      height     - 0 or 0xFFFF if no data present
      exposure   -   100-60000 = 1.00 - 600.00 secs by 0.01
                   60001-60999 = 0.001 - 0.999 secs by 0.001
      packedDate - bits   6-0 = year - 1999 (0 - 127)
                   bits  11-7 = day (1-31)
```

```
                    bits 15-12 = month (1-12)
      packedTime - bits   6-0 = seconds (0-59)
                    bits  7-12 = minutes (0-59)
                    bits 15-13 = hours mod 12 (0-11)
                                 + bit in descriptor can add 12

*/
typedef struct {
      UINT        descriptor;             /* set of bits */
      UINT        height, width;          /* image size */
      UINT        top, left;              /* position in buffer */
      UINT        exposure;               /* exposue time */
      UINT        noExposures;            /* number of exposures added */
      UINT        analogGain;             /* analog gain */
      int         digitalGain;            /* digital gain */
      UINT        focalLength;            /* focal length of scope */
      UINT        aperture;               /* aperture diameter */
      UINT        packedDate;             /* date of image */
      UINT        packedTime;             /* time of image */
      int         ccdTemp;                /* temperature of ccd in 1/100 deg C */
      UINT        siteID;                 /* site id */
      UINT        eGain;                  /* eGain in 1/100th e/ADU */
      UINT        background;             /* background for display */
      UINT        range;                  /* range for display */
      UINT        pedestal;               /* Track and Accumulate pedestal */
      UINT        ccdTop, ccdLeft;        /* position of pixels on CCD array */
} IMAGE_INFO;
```

Request Image Data - Command 0x05, Contains Data

The Request Image Data is sent by the host to the STV to get the Image Data. The format of the data for this command is as follows:

```
UINT  Row;      /* Image Row, 0 thru Image Height - 1 */
UINT  Left;     /* Left Most Pixel, 0 thru Image Width - 1*/
UINT  Length;   /* Number of Pixels, 0 Thru Image Width */
UINT  Buffer;   /* Image Buffer, 0-29 = Flash, 30=Dark, 31=Light */
```

The STV's response is the same command packet with the desired pixel data as UINTs, left most pixel first.

The STV stores additional data in the 201st row (Row above = 200 since zero based). This data is the Track List for Track and Accumulate images or a Mosaic Time List for Mosaic images. The format of these data is shown below:

```
UINT trackList[200]     /* 100 pairs of xOffset, yOffset */

or

UINT mosaicTime[80]     /* 40 pairs of seconds, milliseconds */
```

The Track List contains the X and Y offset of the each image relative to the base image. Positive offsets indicate the individual image was shifted to the right or down then added to the base (accumulated) image. The first pair are always zero since the first image is accumulated with no shift.

The Mosaic Time List contains a pair of 16 bit (WORD) system counters for each image. The counters are synchronous to each other but are separate counters. The first WORD counts up once per

second and the second counts once per millisecond. The data that is recorded is the system time at the end of each exposure (right before readout). For example the first two words are the seconds and milliseconds at the end of the first (upper-left) mosaic image. The second two words are the seconds and milliseconds at the end of the second image (top row, 2nd image from left). The difference in time in milliseconds between the first and second image is:

(mosaicTime[2] - mosaicTime[0])*1000 + (mosaicTime[3] - mosaicTime[1])

The actual date and time that the first image ended is the date/time that's recorded in the Image Info. Using that information and the Mosaic Time list you can determine the time each image ended. Don't ask me why this is so complicated. I think it has something to do with Y2K

Request Compressed Image Data - Command 0x07, Contains Data

The Request Compressed Image Data is sent by the host to the STV to get the Image Data in a compressed format . The format of the data for this request is the same as for the Request Image Data command above. The STV's response is the same command with pixel data, left most pixel first, compressed using the algorithm described in the section below.

6. Host Events

Keyboard Events

Key Down Event - Command 0x08, Contains Data

The Key Down Event is sent by the Host to the STV to simulate the user pressing a key at the STV. The UINT data is the following set of bits:

```
/*

      Rotary Knob Key Patterns

*/
#define LR_ROTARY_DECREASE_PATTERN  0x8000
#define LR_ROTARY_INCREASE_PATTERN  0x4000
#define UD_ROTARY_DECREASE_PATTERN  0x2000
#define UD_ROTARY_INCREASE_PATTERN  0x1000
#define SHIFT_PATTERN               0x0008      /* increases rotary speed when 1 */

/*

      Mode Key Patterns

*/
#define CAL_KEY_PATTERN             0x0100
#define TRACK_KEY_PATTERN           0x0200
#define DISPLAY_KEY_PATTERN         0x0400
#define FILEOPS_KEY_PATTERN         0x0800
#define A_KEY_PATTERN               0x0010
#define SETUP_KEY_PATTERN           0x0020
#define B_KEY_PATTERN               0x0040
#define INT_KEY_PATTERN             0x0080
#define FOCUS_KEY_PATTERN           0x0001
```

```
#define IMAGE_KEY_PATTERN           0x0002
#define MONITOR_KEY_PATTERN         0x0004
```

7. Data Compression Algorithm

This section describes the data compression used by the CPU in response to the Request Compressed Image Data command. The data compression technique is essentially delta compression where each pixel is compared to the previous one sent and if within a small delta then the delta is sent as a byte rather than two bytes. This is a simple compression technique that can give at best 2:1 compression. The CPU's data compression algorithm is as follows:

1) The CPU transmits the first pixel uncompressed as a 16 bit unsigned integer, most significant byte first. It then uses the pixel value as the Base in the following.
2) The CPU calculates Delta = $Pixel_n$ - Base.
3) If -64 <= Delta <= 63 then the CPU transmits a single byte with the most significant bit (b7) clear and Delta in the least significant 7 bits. It then sets Base to the value of $Pixel_n$ and goes to the next pixel and step 2).
4) If -8192 <= Delta <= 8191 (14 bits) then the CPU transmits two bytes. In the first byte it sets the most significant bit (b7) and clears bit b6. In the remaining 6 bits it places the most significant 6 bits of Delta. In the second byte it places the least significant 8 bits of Delta. It then sets Base to the value of $Pixel_n$ and goes to the next pixel and step 2).
5) Otherwise the CPU transmits two bytes. First it calculates Val = $Pixel_n$/4. In the first byte it sets the most significant two bits (b7 & b6), and in the least significant 6 bits it puts the most significant 6 bits of newly calculated Val. In the second byte it places the least significant bits of the newly calculated Val. Finally, it sets Base to Val*4 and goes to the next pixel and step 2).