SBIG STX
HTTP Camera API

24 March 2011

Version 1.00.1

Santa Barbara Instrument Group
147A Castilian Drive
Santa Barbara, CA 93117
Phone: (805) 571-7244 Fax: (805) 571-1147
Web: www.sbig.com EMail: sbig@sbig.com

Introduction

The STX HTTP Camera API allows communication to the camera over Ethernet connections using the standard HTTP protocol. ***The HTTP protocol allows the STX camera to be controlled across a variety of PC platforms without any special driver requirements.***

This API is in addition to the “Classic API” using the SBIG Driver Library.

HTTP Communication

The HTTP communication protocol is a text based communication over standard Ethernet sockets. This standard communication protocol is implemented in several Web Browsers available on PCs, but is also simple enough to be used without a Web Browser. Any programming language with access to a TCP/IP sockets interface can create the messages required to communicate using the HTTP communication protocol.

HTTP Version

The STX implements a web server using the HTTP/1.0 protocol.

HTTP Request Method

All the API calls use the `GET` request method for simplicity. All parameters are appended to the Uniform Resource Identifier (URI) using the standard URI scheme. This is where a question mark (“?”) character is used to indicate the end of the address and the beginning of the parameter list. Then each parameter is separated by an ampersand (“&”) character.

This is an example URI with multiple parameters:

```
http://1.1.1.1/api/ImagerSetSettings.cgi?BinX=2&BinY=2
```

In this example, the API call is to `ImagerSetSettings.cgi`. There are two parameters passed to the function: `BinX`, and `BinY`. Each parameter is assigned a value.

URI Length

The STX interface supports a maximum URI length of 8192 characters.

There is no specific maximum length specified in the HTTP standard. Thus various tools/applications that implement HTTP may not support URIs that are as long as the camera can support.

However, the maximum length of any URI in this API is still considerably shorter than the maximum URI lengths supported in many standards. But the URI length should be a consideration when making any API calls.

URI Character Encoding

The URI has some reserved characters. Any time a reserved character needs to be included in the URI, it must be percent encoded.

The reserved characters and the associated percent encoded value are shown here:

!	*	‘	(	)	;	:	@	&	=	+	$	,	/	?	%	#	[	]	space
%21	%2A	%27	%28	%29	%3B	%3A	%40	%26	%3D	%2B	%24	%2C	%2F	%3F	%25	%23	%5B	%5D	%20

This is an example URI showing the percent encoded values:

```
http://1.1.1.1/api/SetFITSSetting.cgi?ObjectName=California%20Nebula%20%28NGC1499%29
```

This will set the ObjectName parameter to “California Nebula (NGC1499)”.

Example HTTP Conversation

A simple HTTP conversion between a client (PC) and the STX is shown here. This is the transaction that would take place with the following URI (after a TCP/IP socket connection):
`http://1.1.1.1/api/ImagerState.cgi`

Client request:

```
GET /api/ImagerState.cgi HTTP/1.0

```

STX response:

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 3

0

```

See the section *More Example Conversations* for more examples.

HTTP Status Codes

The STX only implements a few of the available HTTP status codes to indicate the result of the request. The used status codes are:

```
200 OK
400 Bad Request
404 Not Found
```

All properly formed requests will result in the "`200 OK`" status code.

Any unrecognized URI will result in the "`404 Not Found`" status code.

Finally, any URI with invalid parameters or other malformed information will result in the "`400 Bad Request`" status code. When the "`400 Bad Request`" is returned, the data portion of the transfer will contain an error number and error text. Possible error numbers are listed with each appropriate API call.

HTTP Content

Any URI that returns text content does so with the content formatted only as plain text. There is no HTML formatting of the response text to simplify parsing on the client. API calls that return multiple values delimit values with a carriage return and line feed (CRLF). The HTTP header will show the Content Type set to "`text/plain`".

Any URI that returns binary type data will show a Content Type set to "`application/octet-stream`" in the HTTP header. After the header is sent the binary data is streamed to the client. Each byte should be saved as part of the binary file. Do not discard CR or LF characters.

HTTP Command Interval

The embedded server in the STX camera is limited in its performance. As such, issuing commands at a very high rate can cause the STX response time to slow down. This is similar to a denial-of-service attack that can occur to web sites on the internet. ***HTTP commands should not be issued to the STX more frequently than one command every 50 milliseconds.***

This command interval results in 20 commands per second, which is still quite a high rate. There is not any functionality that should require this rate of setting or querying parameters. If numerous parameters need to be set or queried in a short period of time, it is better to group the parameters into a single HTTP Request if possible.

If the computer controlling the STX camera is capable of sending commands more frequently than once every 50 milliseconds, a delay should be used to slow down the command interval.

Application Programming Interface (API)

The API is split into four sections. The first three sections contain functions specific to the Imaging CCD, Internal Guide CCD, and External Guide CCD. The last section contains general functions applicable to the entire camera.

Function Description Conventions

Each function can have required and/or optional parameters associated with it. Required parameters are shown in the URI. Optional parameters are shown surrounded by braces “[]”.

Some parameters can be assigned a value. In this case the parameter name should be followed by the equals (“=”) character and then followed by the value to be set to the parameter. Some parameters cannot be assigned a value, but only queried. In this case, there should never be an equals (“=”) character after the parameter’s name.

Imaging CCD

Imager Get Settings

URI: `/api/ImagerGetSettings.cgi?Param1[&Param2]…[&ParamX]`

Available parameters:

Parameter	Description
`BinX`	Binning value for the X axis
`BinY`	Binning value for the Y axis
`CoolerState`	Current cooler state: 0=Off, 1=On
`CCDTemperature`	Current CCD temperature in degrees Celsius
`CCDTemperatureSetpoint`	Current CCD temperature set point in degrees Celsius
`CoolerPower`	Current cooler power level in percent
`CameraXSize`	Width of the camera sensor in unbinned pixels
`CameraYSize`	Height of the camera sensor in unbinned pixels
`ElectronsPerADU`	Gain of the CCD in electrons per A/D unit
`FullWellCapacity`	Full well capacity of the CCD in electrons
`AmbientTemperature`	Current ambient temperature in degrees Celsius
`MaxADU`	Maximum ADU value the CCD will produce
`MaxBinX`	Maximum binning value for the X axis
`MaxBinY`	Maximum binning value for the Y axis
`StartX`	Frame start position for the X axis in un-binned pixels
`StartY`	Frame start position for the Y axis in un-binned pixels
`NumX`	Frame width in un-binned pixels
`NumY`	Frame height in un-binned pixels
`PixelSizeX`	Pixel width in microns
`PixelSizeY`	Pixel height in microns

Description:
This function is used to query any of the parameters listed above. Any number of parameters can be included in a single call to the function.

Returns:
The values for the requested parameters will be returned in the order they were requested.

Errors:
If no valid parameters are included, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001000`	No valid parameter.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/ImagerGetSettings.cgi?CameraXSize&CameraYSize
```

Imager Set Settings

URI: `/api/ImagerSetSettings.cgi?Param1=x[&Param2=x]…[&ParamX=x]`

Available parameters:

Parameter	Description	Valid Values	Default
`BinX`	Binning value for the X axis	`1 to MaxBinX`	`1`
`BinY`	Binning value for the Y axis	`1 to MaxBinY`	`1`

`CoolerState`	Current cooler state	`0 = Off`
		`1 = On`
			`0 = Off`

`CCDTemperatureSetpoint`	CCD temperature set point in degrees C	`-100.0 to 100.0`	`25.0`
`StartX`	Frame start position for the X axis in un-binned pixels	`0 to (CameraXSize – 1)`	`0`
`StartY`	Frame start position for the Y axis in un-binned pixels	`0 to (CameraYSize – 1)`	`0`
`NumX`	Frame width in un-binned pixels	`1 to (CameraXSize – StartX)`	`CameraXSize`
`NumY`	Frame height in un-binned pixels	`1 to (CameraYSize – StartY)`	`CameraYSize`

Description:
This function is used to set any of the parameters listed above. Any number of parameters can be included in a single call to the function.

Returns:
No data is returned.

Errors:
Parameters are parsed in the order they are listed above. If an invalid value is received, a "`400 Bad Request`" error will be generated and no further parsing is performed.

Error Code	Description
`0x80001001`	BinX < 1 or > MaxBin
`0x80001002`	BinY < 1 or > MaxBin
`0x80001003`	StartX < 0 or > (CameraXSize – 1)
`0x80001004`	StartY < 0 or > (CameraYSize – 1)
`0x80001005`	NumX < 1 or > (CameraXSize – StartX)
`0x80001006`	NumY < 1 or > (CameraYSize – StartY)

Invalid parameters are ignored.

Note on `NumX/NumY` parameters. Changing the `StartX/StartY` value impacts the allowed limits of `NumX/NumY`. However, the values of `NumX/NumY` are never changed automatically. The values of `NumX/NumY` are limit checked when an exposure is started and "Bad parameter" is returned if their values are not valid.

Example:
`http://1.1.1.1/api/ImagerGetSettings.cgi?BinX=2&BinY=2&CoolerState=1`

Imager State

URI: `/api/ImagerState.cgi`

Available parameters:
None.

Description:
This function queries the current state of the imaging CCD.

Returns:
Single integer representing the state of the CCD as shown:

Value	State
`0`	Idle
`2`	Exposing
`3`	Reading out the CCD
`5`	Error

Errors:
None.

Example:
`http://1.1.1.1/api/ImagerState.cgi`

Imager Data Binary

URI: `/api/ImagerData.bin`

Available parameters:
None.

Description:
This downloads the binary image data from the camera.

Returns:
A stream of binary image data. Data is 16-bits per pixel, little-endian (low byte first) format. The number of bytes sent to the client is equal to: `(NumX/BinX)*(NumY/BinY)*2`

Errors:
None.

Example:
`http://1.1.1.1/api/ImagerData.bin`

Imager Data FITS

URI: `/api/Imager.FIT`

Available parameters:
None.

Description:
This downloads the binary image data from the camera in FITS format.

Returns:
A stream of FITS format image data. The FITS header contains some user settable parameters. See the FITS Setup function.

Errors:
None.

Example:
`http://1.1.1.1/api/Imager.FIT`

Imager Image Ready

URI: `/api/ImagerImageReady.cgi`

Available parameters:
None.

Description:
Queries the state of the image buffer in the camera.

Returns:
Single integer representing the state of the image buffer in the camera.

Value	State
0	No image available
1	Image is available

Errors:
None.

Example:
`http://1.1.1.1/api/ImagerImageReady.cgi`

Imager Start Exposure

URI: `/api/ImagerStartExposure.cgi?Duration=X&FrameType=Y[&DateTime=Z]`

Available parameters:

Parameter	Description	Valid Values
`Duration`	Duration of the exposure in Seconds	`0.01 to ???`

`FrameType`	Frame type selection	`0 = Dark`
		`1 = Light`
		`2 = Bias`
		`3 = Flat Field`

`DateTime`	Optional exposure Date/Time for FITS header Must be in the format: yyyy-mm-ddThh.mm.ss.sss Where: yyyy = year mm = month dd = day hh = hour mm = minute ss.sss = second to millisecond resolution If not set, the FITS header Date/Time will be set to: `2008-01-01T00:00:00.000`	

Description:
Starts an exposure.

Bias and Flat frame types will only impact FITS headers. In all other aspects Bias is the same as Dark and Flat is the same as Light.

Returns:
No data is returned.

Errors:
If either required parameter is missing, or if the camera is busy, or if an invalid value is sent, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001008`	Camera is busy.
`0x80001009`	Bad parameter.
`0x8000100a`	Parameter(s) missing.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/ImagerStartExposure.cgi?Duration=300&FrameType=1
```

Imager Abort Exposure

URI: `/api/ImagerAbortExposure.cgi`

Available parameters:
None.

Description:
Aborts an exposure in progress. If no exposure is in progress, the abort is ignored.

Returns:
No data is returned.

Errors:
If the abort failed, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001007`	Abort failed.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/ImagerAbortExposure.cgi`

Internal Guide CCD

Guider Get Settings

URI: `/api/GuiderGetSettings.cgi?Param1[&Param2]…[&ParamX]`

Available parameters:

Parameter	Description
`BinX`	Binning value for the X axis
`BinY`	Binning value for the Y axis
`CoolerState`	Current cooler state: 0=Off, 1=On
`CCDTemperature`	Current CCD temperature in degrees Celsius
`CoolerPower`	Currrent cooler power level in percent
`CameraXSize`	Width of the camera sensor in unbinned pixels
`CameraYSize`	Height of the camera sensor in unbinned pixels
`ElectronsPerADU`	Gain of the CCD in electrons per A/D unit
`FullWellCapacity`	Full well capacity of the CCD in electrons
`AmbientTemperature`	Current ambient temperature in degrees Celsius
`MaxADU`	Maximum ADU value the CCD will produce
`MaxBinX`	Maximum binning value for the X axis
`MaxBinY`	Maximum binning value for the Y axis
`StartX`	Frame start position for the X axis in un-binned pixels
`StartY`	Frame start position for the Y axis in un-binned pixels
`NumX`	Frame width in un-binned pixels
`NumY`	Frame height in un-binned pixels
`PixelSizeX`	Pixel width in microns
`PixelSizeY`	Pixel height in microns

Description:
This function is used to query any of the parameters listed above. Any number of parameters can be included in a single call to the function.

Returns:
The values for the requested parameters will be returned in the order they were requested.

Errors:
If no valid parameters are included, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001000`	No valid parameter.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/GuiderGetSettings.cgi?CameraXSize&CameraYSize`

Guider Set Settings

URI: `/api/GuiderSetSettings.cgi?Param1=x[&Param2=x]…[&ParamX=x]`

Available parameters:

Parameter	Description	Valid Values	Default Value
`BinX`	Binning value for the X axis	`1 to MaxBinX`	`1`
`BinY`	Binning value for the Y axis	`1 to MaxBinY`	`1`
`StartX`	Frame start position for the X axis in un-binned pixels	`0 to (CameraXSize – 1)`	`0`
`StartY`	Frame start position for the Y axis in un-binned pixels	`0 to (CameraYSize – 1)`	`0`
`NumX`	Frame width in un-binned pixels	`1 to (CameraXSize – StartX)`	`CameraXSize`
`NumY`	Frame height in un-binned pixels	`1 to (CameraYSize – StartY)`	`CameraYSize`

Description:
This function is used to set any of the parameters listed above. Any number of parameters can be included in a single call to the function.

Returns:
No data is returned.

Errors:
Parameters are parsed in the order they are listed above. If an invalid value is received, a "`400 Bad Request`" error will be generated and no further parsing is performed.

Error Code	Description
`0x80001001`	BinX < 1 or > MaxBin
`0x80001002`	BinY < 1 or > MaxBin
`0x80001003`	StartX < 0 or > (CameraXSize – 1)
`0x80001004`	StartY < 0 or > (CameraYSize – 1)
`0x80001005`	NumX < 1 or > (CameraXSize – StartX)
`0x80001006`	NumY < 1 or > (CameraYSize – StartY)

Invalid parameters are ignored.

Note on `NumX/NumY` parameters. Changing the `StartX/StartY` value impacts the allowed limits of `NumX/NumY`. However, the values of `NumX/NumY` are never changed automatically. The values of `NumX/NumY` are limit checked when an exposure is started and "Bad parameter" is returned if their values are not valid.

Example:
`http://1.1.1.1/api/GuiderGetSettings.cgi?BinX=2&BinY=2&CoolerState=1`

Guider State

URI: `/api/GuiderState.cgi`

Available parameters:
None.

Description:
This function queries the current state of the internal guider CCD.

Returns:
Single integer representing the state of the CCD as shown:

Value	State
`0`	Idle
`2`	Exposing
`3`	Reading out the CCD
`5`	Error

Errors:
None.

Example:
```
http://1.1.1.1/api/GuiderState.cgi
```

Guider Data Binary

URI: `/api/GuiderData.bin`

Available parameters:
None.

Description:
This downloads the binary image data from the camera.

Returns:
A stream of binary image data. Data is 16-bits per pixel, little-endian (low byte first) format. The number of bytes sent to the client is equal to: `(NumX/BinX)*(NumY/BinY)*2`

Errors:
None.

Example:
```
http://1.1.1.1/api/GuiderData.bin
```

Guider Data FITS

URI: `/api/Guider.FIT`

Available parameters:
None.

Description:
This downloads the binary image data from the camera in FITS format.

Returns:
A stream of FITS format image data. The FITS header contains some user settable parameters. See the FITS Setup function.

Errors:
None.

Example:
```
http://1.1.1.1/api/Guider.FIT
```

Guider Image Ready

URI: `/api/GuiderImageReady.cgi`

Available parameters:
None.

Description:
Queries the state of the image buffer in the camera.

Returns:
Single integer representing the state of the image buffer in the camera.

Value	State
`0`	No image available
`1`	Image is available

Errors:
None.

Example:
```
http://1.1.1.1/api/GuiderImageReady.cgi
```

Guider Start Exposure

URI: `/api/GuiderStartExposure.cgi?Duration=X&FrameType=Y[&DateTime=Z]`

Available parameters:

Parameter	Description	Valid Values
`Duration`	Duration of the exposure in Seconds	`0.01 to ???`

`FrameType`	Frame type selection	`0 = Dark`
		`1 = Light`
		`2 = Bias`
		`3 = Flat Field`

`DateTime`	Optional exposure Date/Time for FITS header Must be in the format: yyyy-mm-ddThh.mm.ss.sss Where: yyyy = year mm = month dd = day hh = hour mm = minute ss.sss = second to millisecond resolution If not set, the FITS header Date/Time will be set to: `2008-01-01T00:00:00.000`	

Description:
Starts an exposure.

Bias and Flat frame types will only impact FITS headers. In all other aspects Bias is the same as Dark and Flat is the same as Light.

Returns:
No data is returned.

Errors:
If either required parameter is missing, or if the camera is busy, or if an invalid value is sent, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001008`	Camera is busy.
`0x80001009`	Bad parameter.
`0x8000100a`	Parameter(s) missing.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/GuiderStartExposure.cgi?Duration=11.7&FrameType=1
```

Guider Abort Exposure

URI: `/api/GuiderAbortExposure.cgi`

Available parameters:
None.

Description:
Aborts an exposure in progress. If no exposure is in progress, the abort is ignored.

Returns:
No data is returned.

Errors:
If the abort failed, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001007`	Abort failed.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/GuiderAbortExposure.cgi
```

External Guide CCD

External Guider Get Settings

URI: `/api/ExtGuiderGetSettings.cgi?Param1[&Param2]…[&ParamX]`

Available parameters:

Parameter	Description
`BinX`	Binning value for the X axis
`BinY`	Binning value for the Y axis
`CameraXSize`	Width of the camera sensor in unbinned pixels
`CameraYSize`	Height of the camera sensor in unbinned pixels
`ElectronsPerADU`	Gain of the CCD in electrons per A/D unit
`FullWellCapacity`	Full well capacity of the CCD in electrons
`AmbientTemperature`	Current ambient temperature in degrees Celsius
`MaxADU`	Maximum ADU value the CCD will produce
`MaxBinX`	Maximum binning value for the X axis
`MaxBinY`	Maximum binning value for the Y axis
`StartX`	Frame start position for the X axis in un-binned pixels
`StartY`	Frame start position for the Y axis in un-binned pixels
`NumX`	Frame width in un-binned pixels
`NumY`	Frame height in un-binned pixels
`PixelSizeX`	Pixel width in microns
`PixelSizeY`	Pixel height in microns

Description:
This function is used to query any of the parameters listed above. Any number of parameters can be included in a single call to the function.

Returns:
The values for the requested parameters will be returned in the order they were requested.

Errors:
If no valid parameters are included, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001000`	No valid parameter.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/ExtGuiderGetSettings.cgi?CameraXSize&CameraYSize
```

External Guider Set Settings

URI: `/api/ExtGuiderSetSettings.cgi?Param1=x[&Param2=x]…[&ParamX=x]`

Available parameters:

Parameter	Description	Valid Values	Default Value
`BinX`	Binning value for the X axis	`1 to MaxBinX`	`1`
`BinY`	Binning value for the Y axis	`1 to MaxBinY`	`1`
`StartX`	Frame start position for the X axis in un-binned pixels	`0 to (CameraXSize – 1)`	`0`
`StartY`	Frame start position for the Y axis in un-binned pixels	`0 to (CameraYSize – 1)`	`0`
`NumX`	Frame width in un-binned pixels	`1 to (CameraXSize – StartX)`	`CameraXSize`
`NumY`	Frame height in un-binned pixels	`1 to (CameraYSize – StartY)`	`CameraYSize`

Description:
This function is used to set any of the parameters listed above. Any number of parameters can be included in a single call to the function.

Returns:
No data is returned.

Errors:
Parameters are parsed in the order they are listed above. If an invalid value is received, a “`400 Bad Request`” error will be generated and no further parsing is performed.

Error Code	Description
`0x80001001`	BinX < 1 or > MaxBin
`0x80001002`	BinY < 1 or > MaxBin
`0x80001003`	StartX < 0 or > (CameraXSize – 1)
`0x80001004`	StartY < 0 or > (CameraYSize – 1)
`0x80001005`	NumX < 1 or > (CameraXSize – StartX)
`0x80001006`	NumY < 1 or > (CameraYSize – StartY)

Invalid parameters are ignored.

Note on `NumX/NumY` parameters. Changing the `StartX/StartY` value impacts the allowed limits of `NumX/NumY`. However, the values of `NumX/NumY` are never changed automatically. The values of `NumX/NumY` are limit checked when an exposure is started and “Bad parameter” is returned if their values are not valid.

Example:
```
http://1.1.1.1/api/ExtGuiderGetSettings.cgi?BinX=2&BinY=2&CoolerState=1
```

External Guider State

URI: `/api/ExtGuiderState.cgi`

Available parameters:
None.

Description:
This function queries the current state of the external guider CCD.

Returns:
Single integer representing the state of the CCD as shown:

Value	State
`0`	Idle
`2`	Exposing
`3`	Reading out the CCD
`5`	Error

Errors:
None.

Example:
```
http://1.1.1.1/api/ExtGuiderState.cgi
```

External Guider Data Binary

URI: `/api/ExtGuiderData.bin`

Available parameters:
None.

Description:
This downloads the binary image data from the camera.

Returns:
A stream of binary image data. Data is 16-bits per pixel, little-endian (low byte first) format.
The number of bytes sent to the client is equal to: `(NumX/BinX)*(NumY/BinY)*2`

Errors:
None.

Example:
```
http://1.1.1.1/api/ExtGuiderData.bin
```

External Guider Data FITS

URI: `/api/ExtGuider.FIT`

Available parameters:
None.

Description:
This downloads the binary image data from the camera in FITS format.

Returns:
A stream of FITS format image data. The FITS header contains some user settable parameters. See the FITS Setup function.

Errors:
None.

Example:
`http://1.1.1.1/api/ExtGuider.FIT`

External Guider Image Ready

URI: `/api/ExtGuiderImageReady.cgi`

Available parameters:
None.

Description:
Queries the state of the image buffer in the camera.

Returns:
Single integer representing the state of the image buffer in the camera.

Value	State
0	No image available
1	Image is available

Errors:
None.

Example:
`http://1.1.1.1/api/ExtGuiderImageReady.cgi`

External Guider Start Exposure

URI: `/api/ExtGuiderStartExposure.cgi?Duration=X&FrameType=Y[&DateTime=Z]`

Available parameters:

Parameter	Description	Valid Values
`Duration`	Duration of the exposure in Seconds	`0.01 to ???`

`FrameType`	Frame type selection	`0 = Dark`
		`1 = Light`
		`2 = Bias`
		`3 = Flat Field`

`DateTime`	Optional exposure Date/Time for FITS header Must be in the format: yyyy-mm-ddThh.mm.ss.sss Where: yyyy = year mm = month dd = day hh = hour mm = minute ss.sss = second to millisecond resolution If not set, the FITS header Date/Time will be set to: `2008-01-01T00:00:00.000`	

Description:
Starts an exposure.

Bias and Flat frame types will only impact FITS headers. In all other aspects Bias is the same as Dark and Flat is the same as Light.

Returns:
No data is returned.

Errors:
If either required parameter is missing, or if the camera is busy, or if an invalid value is sent, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001008`	Camera is busy.
`0x80001009`	Bad parameter.
`0x8000100a`	Parameter(s) missing.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/ExtGuiderStartExposure.cgi?Duration=2.52&FrameType=1
```

External Guider Abort Exposure

URI: `/api/ExtGuiderAbortExposure.cgi`

Available parameters:
None.

Description:
Aborts an exposure in progress. If no exposure is in progress, the abort is ignored.

Returns:
No data is returned.

Errors:
If the abort failed, a "`400 Bad Request`" error will be generated.

Error Code	Description
`0x80001007`	Abort failed.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/ExtGuiderAbortExposure.cgi
```

Camera

Filter State

URI: `/api/FilterState.cgi`

Available parameters:
None.

Description:
This function queries the current state of the Filter Selector.

Returns:
Single integer representing the state of the CCD as shown:

Value	State
0	Idle
1	Moving
2	Error

Errors:
None.

Example:
`http://1.1.1.1/api/FilterState.cgi`

Get Filter Settings

URI: `/api/GetFilterSetting.cgi?Param1[&Param2]…[&ParamX]`

Available parameters:

Parameter	Description
`CurrentFilter`	Currently selected filter number
`CurrentFilterName`	Currently selected filter name
`Filter1Name`	Filter 1 Name
`Filter2Name`	Filter 2 Name
`Filter3Name`	Filter 3 Name
`Filter4Name`	Filter 4 Name
`Filter5Name`	Filter 5 Name
`Filter6Name`	Filter 6 Name
`Filter7Name`	Filter 7 Name
`Filter8Name`	Filter 8 Name

Description:
This function queries the current values for the available Filter settings.

Returns:
The values for the requested parameters will be returned in the order they were requested.

Errors:
If no valid parameters are included, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001000`	No valid parameter.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/GetFilterSetting.cgi?CurrentFilter&Filter1Name`

Change Filter

URI: `/api/ChangeFilter.cgi?NewPosition=X`

Available parameters:

Parameter	Description	Valid Values
`NewPosition`	Number of the filter to move to. Can be the currently selected filter.	`0 for no filter or 1 through 8`

Description:
This function moves the Filter Selector to position the selected filter over the camera aperture.

Returns:
No data is returned.

Errors:
If the required parameter is missing, or if the Filter Selector is moving, or if an invalid value is sent, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001009`	Bad parameter.
`0x8000100a`	Parameter(s) missing.
`0x8000100b`	Filter Selector is busy.
`0x8000100c`	Filter Selector not found.
`0x8000100d`	Filter Selector communication error.

Example:
```
http://1.1.1.1/api/ChangeFilter.cgi?NewPosition=0
```

Set Filter Name

URI: `/api/SetFilterName.cgi?Param1=x[&Param2=x]…[&ParamX=x]`

Available parameters:

Parameter	Description	Valid Values	Default Value
`Filter1Name`	Filter Name	Text, 16 characters max	“Empty”
`Filter2Name`			
`Filter3Name`			
`Filter4Name`			
`Filter5Name`			
`Filter6Name`			
`Filter7Name`			
`Filter8Name`			

Description:
This function changes the name of the specified filter.

When these parameters are changed, the new value is programmed into the camera’s non-volatile Flash memory. This allows the camera to remember these parameters after power is removed.

Returns:
No data is returned.

Errors:
None.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/SetFilterName.cgi?Filter1=Luminance
```

Pulse Guide

URI: `/api/PulseGuide.cgi?[DirectionX=W&DurationX=X][DirectionY=Y&DurationY=Z]`

Available parameters:

Parameter	Description	Valid Values

`DirectionX`	X Directional relay to close	`0 = X+`
		`1 = X-`

`DurationX`	Duration of time to keep the X relay closed (in seconds)	`0.01s to 655.35s`

`DirectionY`	Y Directional relay to close	`0 = Y+`
		`1 = Y-`

`DurationY`	Duration of time to keep the Y relay closed (in seconds)	`0.01s to 655.35s`

Description:
Closes relays for the specified time. One call can set either X or Y relays, or both. Each relay will remain closed for the time specified with that relay.

If the requested relay is already activated, the activation time will be reset to the new requested time.

Returns:
No data is returned.

Errors:
If either required parameter is missing, or if an invalid value is sent, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001009`	Bad parameter.
`0x8000100a`	Parameter(s) missing.

Invalid parameters are ignored.

Example:
```
http://1.1.1.1/api/PulseGuide.cgi?DirectionY=1&DurationY=10
```

Is Pulse Guiding

URI: `/api/IsPulseGuiding.cgi`

Available parameters:
None.

Description:
Queries the state of the relays.

Returns:
Single integer representing the state of the relays.

Value	State
`0`	No relay active
`1`	One or more relays active

Errors:
None.

Example:
`http://1.1.1.1/api/IsPulseGuiding.cgi`

Pulse Guide Get Time Remaining

URI: `/api/PulseGuideGetTimeRemaining.cgi?Param1[&Param2]…[&ParamX]`

Available parameters:

Parameter	Description
`XPlus`	X+ Relay Time Remaining
`XMinus`	X- Relay Time Remaining
`YPlus`	Y+ Relay Time Remaining
`YMinus`	Y- Relay Time Remaining

Description:
Queries the time remaining on all the relays.

Returns:
The values for the requested parameters will be returned in the order they were requested.

Errors:
If no valid parameters are included, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001000`	No valid parameter.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/PulseGuideGetTimeRemaining.cgi?XPlus&YMinus`

AO Guide

URI: `/api/AOGuide.cgi?XPosition=X&YPosition=Y`

Available parameters:

Parameter	Description	Valid Values

`XPosition`	AO X Position	`0 to 100`
		`(50 is centered)`

`YPosition`	AO Y Position	`0 to 100`
		`(50 is centered)`

Description:
Moves the AO device to the specified position. Both X and Y positions must be specified.

Returns:
No data is returned.

Errors:
If either required parameter is missing, or if an invalid value is sent, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001009`	Bad parameter.
`0x8000100a`	Parameter(s) missing.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/AOGuide.cgi?XPosition=55&YPosition=42`

AO Status

URI: `/api/AOStatus.cgi`

Available parameters:
None.

Description:
Queries the state of the AO device.

Returns:
Single integer representing the state of the AO.

Value	State
`0`	Idle
`1`	Moving

Errors:
None.

Example:
`http://1.1.1.1/api/AOStatus.cgi`

Get FITS Settings

URI: `/api/GetFITSSetting.cgi?Param1[&Param2]…[&ParamX]`

Available parameters:

Parameter	Description
`ObjectName`	Object Name field
`Observer`	Observer field
`Telescope`	Telescope field
`FL`	Focal Length field
`Aperture`	Aperture Diameter field
`Area`	Aperture Area field

Description:
This function queries the current values for the available FITS header fields.

Returns:
The values for the requested parameters will be returned in the order they were requested.

Errors:
If no valid parameters are included, a “`400 Bad Request`” error will be generated.

Error Code	Description
`0x80001000`	No valid parameter.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/GetFITSSetting.cgi?ObjectName&FL`

Set FITS Settings

URI: `/api/SetFITSSetting.cgi?Param1=X[&Param2=X]…[&ParamX=X]`

Available parameters:

Parameter	Description	Valid Values	Default Value
`ObjectName`	Object Name field	Text, 67 characters max	"Object Description"
`Observer`	Observer field	Text, 67 characters max	"STX Camera Operator"
`Telescope`	Telescope field	Text, 67 characters max	"Telescope Description"
`FL`	Focal Length field	Floating point number	2000.00
`Aperture`	Aperture Diameter field	Floating point number	200.00
`Area`	Aperture Area field	Floating point number	25000.00

Description:
Sets any of the available FITS header fields. Used only for FITS files. All text parameters may contain any ASCII characters with values from decimal 32 through 126 (hexadecimal 20 through 7E). This is the same requirement as in the FITS standard.

When these parameters are changed, the new value is programmed into the camera's non-volatile Flash memory. This allows the camera to remember these parameters after power is removed.

Returns:
No data is returned.

Errors:
None.

Invalid parameters are ignored.

Example:
`http://1.1.1.1/api/SetFITSSetting.cgi?ObjectName=M27`

Description

URI: `/api/Description.cgi`

Available parameters:
None.

Description:
Queries the camera model number.

Returns:
Text string containing the camera model number, e.g. “SBIG STX-16803”.

Errors:
None.

Example:
`http://1.1.1.1/api/Description.cgi`

Version Numbers

URI: `/api/VersionNumbers.cgi`

Available parameters:
None.

Description:
Queries the camera version numbers.

Returns:
Text string containing the camera version numbers, in the following order:

Firmware Version
Gate Array Version
Imaging ROP Version
Tracker ROP Version
HTTP API Version

Errors:
None.

Example:
`http://1.1.1.1/api/VersionNumbers.cgi`

Example Source Code

These examples are all written in Microsoft Visual Studio 2008 using the Visual C# Console Application template.

To use these examples, create a new project called "`STXWebAPITest`" using the template. Replace all of the text in the `Program.cs` file with the text below. Finally change the IP address in the example from "`1.1.1.1`" to the IP address of your camera.

Reading a Value

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;

namespace STXWebAPITest
{
    class Program
    {
        static void Main(string[] args)
        {
            HttpWebRequest Request;
            String CameraDescription;

            try
            {
                // prepare the web page we will be asking for
                Request = (HttpWebRequest)WebRequest.Create("http://1.1.1.1/api/Description.cgi");
            }
            catch
            {
                Console.Write("Error creating request.\n");
                Request = null;
            }

            // execute the request
            if (Request != null)
            {
                try
                {
                    HttpWebResponse Response = (HttpWebResponse)Request.GetResponse();
                    StreamReader sr = new StreamReader(Response.GetResponseStream());
                    CameraDescription = sr.ReadLine();
                    Console.WriteLine(CameraDescription);
                    sr.Close();
                }
                catch
                {
                    Console.Write("Error getting responce.\n");
                }
            }

            Console.Write("Press any key to continue.\n");
            Console.ReadKey();
        }
    }
}
```

Downloading an Image

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;

namespace STXWebAPITest
{
    class Program
    {
        static void Main(string[] args)
        {
            HttpWebRequest Request;

            try
            {
                // prepare the web page we will be asking for
                Request = (HttpWebRequest)WebRequest.Create("http://1.1.1.1/api/ImagerData.bin");
            }
            catch
            {
                Console.Write("Error creating request.\n");
                Request = null;
            }

            // execute the request
            if (Request != null)
            {
                try
                {
                    HttpWebResponse Response = (HttpWebResponse)Request.GetResponse();
                    Stream ImageStream = Response.GetResponseStream();
                    Stream myFileStream = File.Create("ImagerData.bin");

                    try
                    {
                        int Length = 16384; // 16384 bytes is the most the STX will send at once
                        Byte[] buffer = new Byte[Length];
                        int bytesRead = ImageStream.Read(buffer, 0, Length);
                        while (bytesRead > 0)
                        {
                            myFileStream.Write(buffer, 0, bytesRead);
                            bytesRead = ImageStream.Read(buffer, 0, Length);
                        }
                    }
                    catch
                    {
                        Console.Write("Error saving file.\n");
                    }

                    myFileStream.Close();
                    ImageStream.Close();
                }
                catch
                {
                    Console.Write("Error getting responce.\n");
                }
            }

            Console.Write("Press any key to continue.\n");
            Console.ReadKey();
        }
    }
}
```

Full Featured Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;

namespace STXWebAPITest
{
    class Program
    {
        static void Main(string[] args)
        {
            TakeImage();

            Console.Write("Press any key to continue.");
            Console.ReadKey();
        }

        static void TakeImage()
        {
            bool RetVal = true;
            String Response = "";

            if (RetVal)
            {
                Console.WriteLine("Setting imager parameters.");
                RetVal = PerformWebRequest("ImagerSetSettings.cgi?StartX=0&StartY=0&NumX=640&NumY=480", ref Response);
            }

            if (!RetVal)
            {
                Console.WriteLine("Error setting parameters.");
                return;
            }
            else
            {
                Console.WriteLine("Starting exposure.");
                RetVal = PerformWebRequest("ImagerStartExposure.cgi?Duration=10&FrameType=1", ref Response);
            }

            if (!RetVal)
            {
                Console.WriteLine("Error starting exposure.");
                return;
            }
            else
            {
                int ImagerState = 5;    // 5 is an error condition
                Console.WriteLine("Waiting for exposure to finish...");

                do
                {
                    RetVal = PerformWebRequest("ImagerState.cgi", ref Response);
                    if (RetVal)
                        ImagerState = Convert.ToInt32(Response);
                }
                while ((ImagerState != 0) && (ImagerState != 5) && RetVal); // 0 is Idle, 5 is an error

                if (ImagerState == 5)   // Got an error condition
                {
                    Console.WriteLine("Imager State Error.");
                    return;
                }
            }

            if (RetVal)
            {
                Console.WriteLine("Setting FITS parameters.");
                RetVal = PerformWebRequest("SetFITSSetting.cgi?ObjectName=M27&Observer=SBIG&Telescope=RC10", ref Response);
            }

            if (!RetVal)
            {
                Console.WriteLine("Error setting FITS parameters.");
                return;
            }
            else
            {
                Console.WriteLine("Getting image FITS file.");
                RetVal = PerformWebRequest("Imager.FIT", "Imager.FIT");
            }

            if (!RetVal)
            {
                Console.WriteLine("Error getting Imager.FIT.");
                return;
            }
            else
                Console.WriteLine("All done.");
        }

        const String HTTPAddressStart = "http://1.1.1.1/api/";

        // Perform Web Request and return response in a string
        // This function is overloaded.
        static bool PerformWebRequest(String Address, ref String Response)
        {
            HttpWebRequest Request;

            try
            {
                // prepare the web page we will be asking for
                Request = (HttpWebRequest)WebRequest.Create(HTTPAddressStart + Address);
            }
            catch
            {
                // Something failed
                Response = "";
                return false;
            }

            try
            {
                // Get response
                HttpWebResponse WebResponse = (HttpWebResponse)Request.GetResponse();

                if (WebResponse.StatusCode == HttpStatusCode.OK)
                {
                    // Read response text
                    StreamReader sr = new StreamReader(WebResponse.GetResponseStream());
                    Response = sr.ReadToEnd();
                    sr.Close();
                }
                else
                {
                    // Camera returned 400 or 404
                    Response = "";
                    return false;
                }
            }
            catch
            {
                // Something failed
                Response = "";
                return false;
            }

            return true;
        }

        // Perform Web Request and write the response to a file
        // This function is overloaded.
        static bool PerformWebRequest(String Address, String FileName)
        {
            HttpWebRequest Request;

            try
            {
                // prepare the web page we will be asking for
                Request = (HttpWebRequest)WebRequest.Create(HTTPAddressStart + Address);
            }
            catch
            {
                // Something failed
                return false;
            }

            try
            {
                // Get response
                HttpWebResponse WebResponse = (HttpWebResponse)Request.GetResponse();

                if (WebResponse.StatusCode == HttpStatusCode.OK)
                {
                    // Get data and dump into the file
                    Stream ImageStream = WebResponse.GetResponseStream();
                    Stream myFileStream = File.Create(FileName);
                    try
                    {
                        int Length = 16384; // 16384 bytes is the most the STX will send at once
                        Byte[] buffer = new Byte[Length];
                        // Read data block, then write data block
                        int bytesRead = ImageStream.Read(buffer, 0, Length);
                        while (bytesRead > 0)
                        {
                            myFileStream.Write(buffer, 0, bytesRead);
                            bytesRead = ImageStream.Read(buffer, 0, Length);
                        }
                        // All done, clean up
                        myFileStream.Close();
                        ImageStream.Close();
                    }
                    catch
                    {
                        // Something failed, close these and return
                        myFileStream.Close();
                        ImageStream.Close();
                        return false;
                    }
                }
                else
                {
                    // Camera returned 400 or 404
                    return false;
                }
            }
            catch
            {
                // Something failed
                return false;
            }

            return true;
        }
    }
}
```

More Example Conversations

Multiple Parameter Get

Client request:

```
GET
/api/ImagerGetSettings.cgi?MaxADU&MaxBinX&MaxBinY
&PixelSizeX HTTP/1.0
```

STX response:

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 19

65535
9
9
9.00
```

No Valid Parameter Error

Client request:

```
GET /api/ImagerGetSettings.cgi HTTP/1.0
```

STX response:

```
HTTP/1.0 400 Bad Request
Content-Type: text/plain
Content-Length: 33

0x80001000
No valid parameter.
```

Successful Conversion with No Data

Client request:

```
GET /api/ImagerSetSettings.cgi?BinX=1 HTTP/1.0
```

STX response:

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 0
```

Revision History

This section details the recent changes to this specification since the initial release.

Changes Incorporated in Version 1.00.1:

- Added Revision History.
- Added HTTP Command Interval restriction.