ISSI LC-1S

CCTV Lens Controller

Application Program Interface

Programmers Reference Manual

Version 3.0.9 (FW 3.0.6)

Notice

Contact Information: Innovative Scientific Solutions Inc.
7610 McEwen Road Dayton, OH 45459 Ph.: (937) 630-3012 Fax: (937) 630-3015
Website: www.innssi.com, Email : issi-sales@innssi.com

ISSI LC-1S API

API based on UDP Protocol, port 1337/UDP.

1. Commands

Action: Get controller f/w version
Command: “ver” (hex: 766572)
Returns: “*ISSI LC-1S=1.x.y (s/n:19xxx)*”

Action: Change controller IP address to 192.168.1.2
Command: “*ChangeIP=192.168.1.2*” (hex: 4368616e676549503d3139322e3136382e312e32)
Returns: “*IPchanged*”

Action: Get current motors values
Command: “*Current*” (hex: 43757272656e74)
Returns: “*Current=XXX,YYY,ZZZ*” Where XXX – zoom position value, YYY- focus position value, ZZZ – iris position value.

Action: Move Zoom Narrow during X ms
Command: “*ZoomN=X*”
Returns: “*Zoom=Y*”, where Y – current value of Zoom motor

Action: Move Zoom Wide during X ms

Command: “*ZoomW=X*”

Returns: “*Zoom=Y*”, where Y – current value of Zoom motor

Action: Move Focus Far during X ms

Command: “*FocusF=X*”

Returns: “*Focus=Y*”, where Y – current value of Focus motor

Action: Move Focus Near during X ms

Command: “*FocusN=X*”

Returns: “*Focus=Y*”, where Y – current value of Focus motor

Action: Move Iris Open during X ms

Command: “*IrisO=X*”

Returns: “*Iris=Y*”, where Y – current value of Iris motor

Action: Move Iris Close during X ms

Command: “*IrisC=X*”

Returns: “*Iris=Y*”, where Y – current value of Iris motor

Command *findLimits needs to be issued to determine lens motor limits first.*

Action: Set zoom motor value to X

Command: “*setZoom=X*”

Returns: “*zoomDone*”, it may take a while to position the motors.

Command *findLimits* needs to be issued to determine lens motor limits first, otherwise returns *errorZoom*.

Action: Set focus motor value to X

Command: “*setFocus=X*”

Returns: “*focusDone*”, it may take a while to position the motors.

Command *findLimits* needs to be issued to determine lens motor limits first, otherwise returns *errorFocus*.

Action: Set iris motor value to X

Command: “setIris=X”

Returns: “irisDone”, it may take a while to position the motors.

Command *findLimits* needs to be issued to determine lens motor limits first, otherwise returns *errorIris*.

Action: Stop all motors (during setFocus, setIris or setZoom)

Command: “*setStop*”

Returns: same as for ‘Current” command

Action: Motors limits detection (for all tree motors)

Command: “*findLimits*” (or “*FindLimits*”, old)

Returns: in separate packets: *“focusMin=XXX” “focusMax=XXX”, “irisMin=XXX”, “irisMax=XXX”, “zoomMin=XXX”, “zoomMax=XXX”*

This command needs to be issued first before using setZoom, setFocus or setIris commands. Limits will be stored in internal LC-1S memory and automatically recalled after each device restart.

Action: Print motor limits detected previously (for all tree motors)

Command: “*getLimits*”

Returns: in separate packets: , *“zoomMin=XXX”, “zoomMax=XXX”, “focusMin=XXX” “focusMax=XXX”, “irisMin=XXX”, “irisMax=XXX”*

Action: Set Iris mode - Motorized Iris(X=2, default), DC Iris(X=1) iris or Video Iris(X=0). **Command**: "*setIrisMode=X*"

Returns: "*mode=X*"

Action: Move DC Iris to close

Command: "*DIrisC=X*"

Returns: "*DIrisC=OK*", DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move DC Iris to open

Command: "*DIrisO=X*"

Returns: "*DIrisO=OK*", DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move DC Iris to close faster

Command: "*DIrisCX2=X*"

Returns: "*DIrisCX2=OK*", DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move DC Iris to open faster

Command: "*DIrisOX2=X*"

Returns: "*DIrisOX2=OK*", DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move Video Iris to close

Command: "*VIrisC=X*"

Returns: “*VIrisC=OK*”, DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move Video Iris to open
Command: “*VIrisO=X*”
Returns: “*VIrisO=OK*”, DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move Video Iris to close faster
Command: “*VIrisCX2=X*”
Returns: “*VIrisCX2=OK*”, DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Move Video Iris to open faster
Command: “*VIrisOX2=X*”
Returns: “*VIrisOX2=OK*”, DC Iris lenses have no potentiometer, there is no feedback with the actual position. Where X is a midpoint, please see manual.

Action: Set nickname for LC (stored in LC memory)
Command: “*setNAME=XXXXXXX*” where XXXXXXX is 7-symbols name
Returns: no answer

Action: Get nickname for LC (stored in LC memory)
Command: “*getNAME*”
Answer: “*NAME=XXXXXX*”

Action: soft restart LC-2
Command: “*Reboot*”
Returns: “*OK*”

2. Examples

Python3

```
-----------------------------------------------------------------

import socket
import time

# LC2 IP address
UDP_IP = "192.168.2.251"
UDP_PORT = 1337

print ("ISSI :: LC-1S Python Script Example\n")

# Command list
MESSAGES = ["ver", "IrisMode=2", "Current"]

# UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP

sock.setblocking(0)
sock.settimeout(0.05)
counter = 0
# Loop on command list
for i in range(len(MESSAGES)): # LC2 command loop
        print ("#",i+1,"Sent:", MESSAGES[i])
        start = time.time()
        sock.sendto(MESSAGES[i].encode(), (UDP_IP, UDP_PORT))
        try:
                while True:
                        data, addr = sock.recvfrom(1024)
                        if not data: break
                        print ("Received:", data)
                        end = time.time()
                        print(end - start, " seconds")
        except socket.error:
        print("")
        time.sleep(0.15)
```
