InternationalLight
TECHNOLOGIES
10 Technology Drive
Peabody, MA 01960
Ph: 978-818-6180
Fax: 978-818-6181
Web: www.intl-lighttech.com

ILT5000 & ILT1000 Command & Control
CLI, API, Theory of Operation & Labview Code

The ILT5000 and ILT1000 were designed to accept commands over a serial port. Both devices use the same theory of operation. The ILT5000 works with all ILT1000 commands plus some extended features and commands. The extensive API included in this manual can be accessed to perform many of the task found in ILT's BAR, TREND and METER applications using Labview or by interfacing with any standard terminal program (hyperterminal, putty, MAC terminal window, etc).

The ILT1000 devices can be used on pc's running Windows XP, 7 or 8 as well as on MAC computers. The ILT5000 devices cannot be used on pc's running Windows XP.

Table of contents

CLI:
CLI is a very basic command line interface program contained within Datalight and Datalight II.

1. Starting CLI: Double click on the CLI icon located the DataLight folder or on the desktop (depending of the rev of software you are using)

When run, CLI will scan through all available ports noting "No device found at COM port #" for each port until it finds a meter. CLI will automatically detect the first available device and present a CLI interface. If you wish to control additional devices, running another instance of CLI will open up another CLI window to control the next device found.

2. Command List: When CLI opens you will receive a prompt "Type help for a list of device commands Type exit or quite to close the console". Type "help" and press return. This will display a list of the most commonly used commands for your version of firmware.

help screen

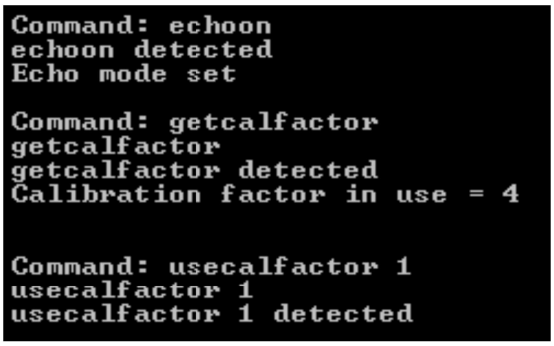
This list also serves as the documented API for the device. A more comprehensive version of the API is shown below, starting on page 5**.**

3. Display options: CLI opens in "**echooff**" mode, where all responses are numerical. Type **echoon** to swith to from numerical to text. The command prompt with a flashing curser will appear.

4. Issue commands: To use CLI simply type any of the commands shown in the help screen (or from the API) and hit enter.

5. Taking a light measurement readings with CLI:
CLI allows the user to verify their calibration factor and take a single reading or log data into the internal memory.

5A Calibration factor: Calibrated ILT1000/ILT5000 units are shipped with the calibration factor pre-programmed. CLI can access 20 calibration factor channels used by the meter to store calibrations.



1. getcalfactor: To determine which calibration factor is currently in use type "getcalfactor" and hit return. CLI will respond with the answer, Calibration factor in use = # (# is channel 1 thru 20).

2. usecalfactor: If the channel noted is not the correct channel enter "usecalfactor #".
For example if you have a UVA filter with a cal factor in channel 1 and a Y VIS light filter with a cal factor in channel two. You must first assure your UVA filter is on the ILT1000 or ILT5000 sensor, Type "usecalfactor 1" and hit enter.
(Note: The calibration factor channel and CLI do not save or display the units. For example, the UVA readings are typically in W/cm2 or W/m2 and the VIS light readings are typically in fc or lux. Please refer to the calibration certificate to determine the units of measurement)

3. setcalfactor: The ILT1000/ILT5000 allows the user to enter their own calibration factor. User must input calibration factor description (up to 100 characters), the current-to-irradiance multiplier and saturation current in microamps. Example: setcalfactor 5 calfact5 3.28e-4 50
(Note: only experienced users should attempt self calibration /setcalfactor)

5B. Taking a single reading: To take a single light level reading type "getirradiance" and hit enter. A single reading in scientific notation will appear, for example 5 mW/cm2 will be displayed as 5.00e-03.

5c. Logging data: The format for logging data is:
startlogdata"space"bitmask sum"space"time in millisecond intervals"space"0 (or epoch time)
ie. startlog data 32 1000 0

1: Bitmask. To create a bitmask, select the items from the bitmask options below, and sum the total for the selected bitmap values . For example if you wish to log **4=**Detector Current (picoamps) and **32=**Calibrated light level (see getirradiance) you use a bitmap of 36 (4 + 32) Ie startlogdata **36**

Recorded Value Indicator bitmask as follows:
1=Optical Density (x100)
2=Percent Transmission (x10)
4=Detector Current (picoamps)
8=Detector Voltage (microvolts)
16=Device board temperature (degrees F)

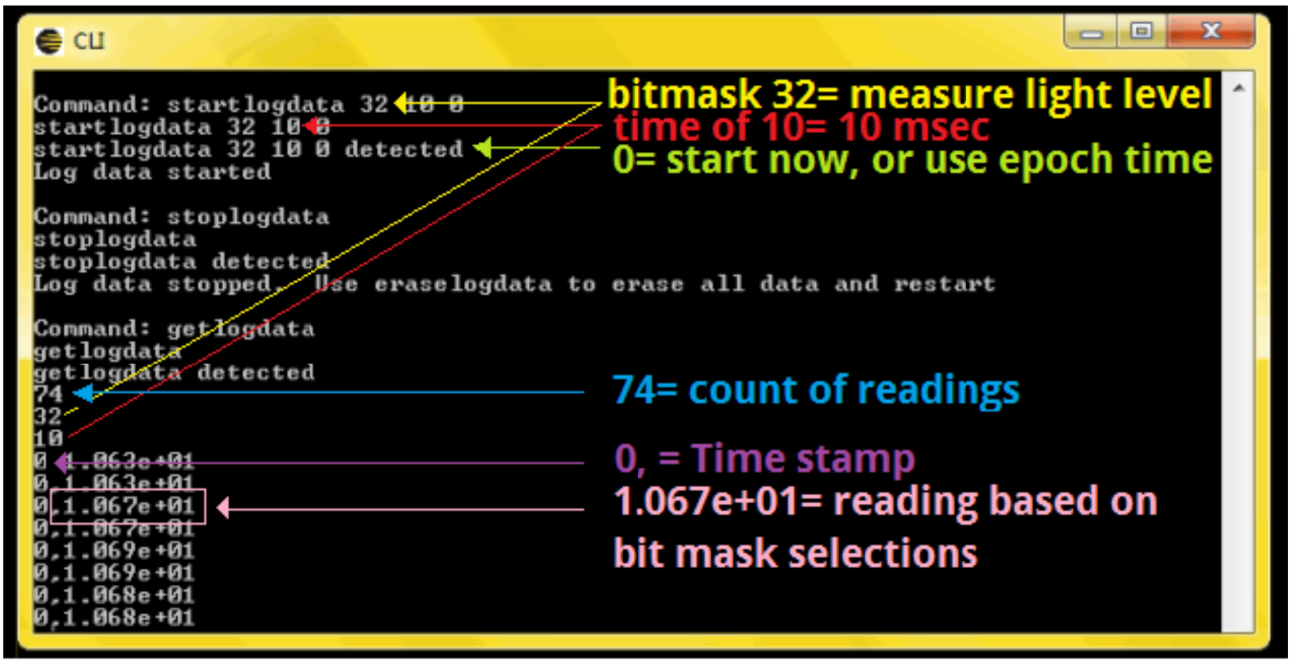
32=Calibrated light level (see getirradiance)

2. Recording interval: Readings are taken at 1 msec/ 1000 reading per second intervals. The record interval is the amount of msec between storing a value. A setting of 1 would save 1000 readings per second. To read once per second use interval of 100 ie startlogdata 36 ***100.***
Note: for firmware versions 2.0.0.2 a 1 = 1second delay between logging; a value of 3600 would indicate a 3600 second / 1 hour delay. The maximum value is 86400 / 1 day.

3. Start time: the last digit in the startlogdata is the start time. Use a ***0*** to begin recording immediately. Ie startlogdata 36 100 ***0***
Epoch time is used to delay the start time. Enter the epoch time [seconds since 1970] This is "Unix epoch time", with most application coding environments having a mechanism to convert a date-time structure to epochtime. There are easy to use converters online: http://www.epochconverter.com/

If done correctly, the terminal will display: log data started.

Example startlogdata 32 10 0.



5D. View readings during testing: To display the results without interrupting readings, type: getlogdata.

5E. Finish testing and veiw results: To stop the measurements and obtain readings, type: stoplogdata and then getlogdata

5F. Erase old data: The ILT1000/ILT5000 can only store one set of test data. Before starting a new session you will need to erase the old session. To erase the readings stored in memory type: eraselogdata.

API

International Light technologies has created a simple text-based API for communication between the ILT1000/ILT5000 and user applications, or a basic terminal server. The API is documented below. Please Note; you can also obtain a sample of the more commonly used API commands by type help in the CLI menu. While we have provided you with a large portion of the API, most customers will only use a small sub-set of the API's vast capabilities.

Documentation Convention

The Datalight API table below contains two columns. In the first column is the API/command name. In the second column is the API/command's definition. The definition contains several elements and convention as follows.

- Actual command and responses will use the Courier New font.
- Command parameters are identified in [brackets], within the "Syntax" description. The brackets themselves are not used in the command syntax, as is illustrated by the "Example command" listed for any command with parameters.
- The "Normal return value(s)" and "Error return value(s)" information is pertinent to the echooff mode of operation. (see echoon and echooff commands in the table below). When using echoon mode, return values are self explanatory.
- An N/A for an "Error return value(s)" indicates that errors are not returned and not expected for this command.
- "Example return" is listed, on some commands, to provide an example of the output. In this field, any text following a dash ("-"), as well as the dash itself,are not part of the actual output and only listed to help describe the output.
- All standard commands are listed alphabetically(followed by programming commands which have an * before the name.)
- The optional "Persist through power-cycle" field will indicate whether or not the configuration is stored in flash memory and "sticks" through power cycles.
- Note that all commands are case sensitive, and always **all-lowercase**.

Product Generations

The Datalight API supports all generations of the ILT1000/ILT5000 and firmware. If you are having trouble with the API or any of the applications in DataLight or DataLight II, please email ilsales@intl-lighttech.com. Be sure to include your model and serial number, as well as the revision of the software and firmware you are using. Software and firmware updates can be downloaded from the ILT website: http://www.intl-lighttech.com/support/software.

API List:

`echoon`	Syntax: `echoon` What is does: This command enters a verbose mode whereby contextual help is echoed back for each command completion. The mode is useful when interacting with the RTGOM from a terminal server or from the RTGOM-CLI program. Normal return value: 0 on success Error return value(s): N/A Persist through power-cycle:No
`eraselogdata`	Syntax: `eraselogdata` What it does: This command erases any log data that has been stored in the RTGOM flash memory. See also `startlogdata` and `stoplogdata`. Normal return value(s): 0 on success Error return value(s): -500 if logging is currently active (must use stoplogdata first) -501 error erasing the data from flash Persist through power-cycle:Yes
`get100perc`	Syntax: `get100perc` What it does: This command returns the sensor voltage at the time the 100% value was set with `set100perc`. Normal return value(s): Voltage for 100% transmission, in microvolts

	Error return value(s): -500 if the 100% value was never set
`getambienttemp` `(Gen2 only)`	Syntax: `getambienttemp` What it does: This command returns the ambient temperature as sensed by the device, times 100. This differs from `gettemp` which returns the internal temperature of the microcontroller. Normal return value(s): Temperature in degrees F (x100). Error return value(s): N/A
`getcurrent`	Syntax: `getcurrent` What it does: This command returns the sensor current in picoamps. Normal return value(s): Current in picoamps Error return value(s): -500 if there is a voltage saturation, indicating the reading must be discarded
`getdarkmode`	Syntax: `getdarkmode` What it does: This command returns the dark mode currently in use by the device. The dark mode can be NO DARK (there is no consideration for photodiode dark current), FACTORY DARK (the dark current set at the factory), or USER DARK (the dark current set with `setuserdark`). Return value(s): 0 = NO DARK 1 = FACTORY SET 2 = USER SET Error return value(s): N/A

`getdatetime`
`(Gen2 only)`
	Syntax:
	`getdatetime`
	What it does:
	This command returns the real time clock's date and time in formats as shown below. This format includes a typical date and time format (mm/dd/yyyy hh:mm:ss), followed by the Epoch time (seconds since 1970). Note that the device is programmed to UTC/GMT time and, as a result, might return a time that does not match the local timezone. Modern computer programming languages will correctly convert the Epoch time data to local time when saving the timestamped log data. This is the case with the Data Logger software.
	Return value(s):
	Real time clock time as shown below
	Error return value(s):
	N/A
	Example return:
	`12/05/2013 19:02:05 1386270125`

`getfactorydark`	Syntax:
	`getfactorydark`
	What it does:
	This command returns the dark voltage at the various transimpedance amplifier gain stages, as set at the factory.
	Normal return value(s):
	The x1 dark voltage in microvolts, followed by a space, followed by the x101 dark voltage in microvolts
	Error return value(s):
	-500 if the factory dark voltage has not been set
	Example return:
	`12756 9234`

`getfwversion`	Syntax: `getfwversion`

	What it does: This command returns the firmware version running on the device. Normal return value(s): Device firmware version Error return value(s): N/A Example return: `2.3.0.5`

`getgeneration`	Syntax:
	`getgeneration`
	What it does:
	This command returns the generation of the device.
	Normal return value(s):
	-999 for first generation products that did not have this command defined at time of release
	1 for first generation products
	2 for second generation products (programmable Rf, realtime clock, ambient temp sensor)
	Error return value(s):
	N/A
	Example return:
	`2`

`getirradiance`	Syntax:
	`getirradiance`
	What it does:
	This command returns the irradiance value in user-defined units. Irradiance is determined based on the following priority list:
	1. If a calibration factor is in use (see `usecalfactor`), this current-to-irradiance multiplier value is used. This is the most common method to derive irradiance.
	2. If no calibration factor is in use, or the calibration factor has its multiplier value set to 65535 (0xFFFF), the custom calibration table set with either `setsimpleirrcal` or multiple calls to `setirrdatapoint` is used.
	Normal return value(s):
	Irradiance value in user-defined units, x 1000
	Error return value(s):
	-500 if no irradiance calibration data has been set
	-501 if the detector current is out of range for the current-to-irradiance curve (case #2 above)
	-502 if there is a voltage saturation, indicating the reading must be discarded
	Example return (for an irradiance value of 73.798):
	`73798`

`getlogdata`	Syntax: `getlogdata` What it does: This command returns the log data stored in flash memory. This command can be run during an active logging session (and datalogging will continue) or after a session is stopped with `stoplogdata`. The command will first output three values (total number of values, an integer indicating what information was logged, the logging period in seconds), followed by lines of comma-delimited data with date-time-stamp (in seconds since 1970 or “Unix epoch time”) followed by all recorded values for that date-time-stamp. Normal return value(s): Total number of Date-Time-Stamp + Value pairs logged Recorded Value Indicator bitmask as follows: 1=Optical Density (x100) 2=Percent Transmission (x10) 4=Detector Current (picoamps) 8=Detector Voltage (microvolts) 16=Device board temperature (degrees F) 32=Calibrated Irradiance (see `getirradiance`) Logging Period (in Seconds)

	Seconds Since 1970, value #1, value #2, value #n Seconds Since 1970, value #1, value #2, value #n Seconds Since 1970, value #1, value #2, value #*n* Error return value(s): -500 if no log data present Notes on returned data values: • If the 100% is not set, both Optical Density and Percent Transmission will return 0. • Log data does not support negative Optical Density values • Calibrated Irradiance will return 0 if there is no calibration data Example return (Notes after the '-' are for documentation purposes and not returned): `5` – total time-stamp + value pairs `4` – detector current, in picoamps, is what was logged `60` – period in seconds (sample every minute) `1378738200, 159564` – 09 Sep 2013 14:50:00 GMT, 159.564 nanoamps `1378738260, 134657` – 09 Sep 2013 14:51:00 GMT, 134.657 nanoamps `1378738320, 145671` – 09 Sep 2013 14:52:00 GMT, 145.671 nanoamps `1378738380, 174801` – 09 Sep 2013 14:53:00 GMT, 174.801 nanoamps `1378738440, 163714` – 09 Sep 2013 14:54:00 GMT, 163.714 nanoamps

`getmodelname`	Syntax:
	`getmodelname`
	What it does:
	This command returns the model name of the device.
	Normal return value(s):
	Device model name
	Error return value(s):
	N/A
	Example return:
	`ILT5000`

`getod`	Syntax:
	`getod`
	What it does:
	This command returns the optical density, multiplied by 100. Because optical density is a relative measurement, this command requires that the 100% (0.00 OD) setting is established with `set100perc`.
	Normal return value(s):
	Optical density x 100
	Error return value(s):
	-500 if the 100% value has not been previously set with `set100perc`
	Example return (for an optical density of 1.07):
	`107`

`getserialnumber`	Syntax:
	`getserialnumber`
	What it does:
	This command returns the serial number of the device. The serial number is stored in one-time-programmable memory at the time of manufacture.
	Normal return value(s):
	Device serial number
	Error return value(s):
	-500 device serial number has not been set
	Example return:
	`10054201208230245`

`gettemp`	Syntax: `gettemp` What it does: This command returns the internal temperature of the device microcontroller. Note that, while the detector temperature can be loosely inferred from this, this is not equivalent to the detector temperature. The return value is in degrees F, with any units conversion performed by the application software. Normal return value(s):

	device microcontroller temperature, in degrees F Error return value(s): N/A Example return: `107`

`gettrans`	Syntax:
	`gettrans`
	What it does:
	This command returns the percent transmission, multiplied by 10. Because percent transmission is a relative measurement, this command requires that the 100% (0.00 OD) setting is established with `set100perc`.
	Normal return value(s):
	Percent Transmission x 10
	Error return value(s):
	-500 if the 100% value has not been previously set with `set100perc`
	Example return (for a percent transmission of 67.3):
	`673`

`getuserdark`	Syntax:
	`getuserdark`
	What it does:
	This command returns the dark voltage at the various transimpedance amplifier gain stages, as set by the user with `setuserdark`.
	Normal return value(s):
	The x1 dark voltage in microvolts, followed by a space, followed by the x101 dark voltage in microvolts
	Error return value(s):
	-500 if the factory dark voltage has not been set
	Example return:
	`13014 9832`

`getvoltage`	Syntax:
	`getvoltage`
	What it does:
	This command returns the voltage output of the transimpedance amplifier, after it is passed through the automatic-gain-control circuit.
	Normal return value(s):
	Detector voltage, in microvolts
	Error return value(s):
	N/A
	Example return (for 2.415896 volts):
	`2415896`

`set100perc`	Syntax:
	`set100perc`
	What it does:
	This command reads the detectors voltage level (and by inference its current and irradiance level) and sets this as the 100% value to use in Optical Density and %Transmission calculations.
	Normal return value(s):
	The 100% voltage value, in microvolts
	Error return value(s):
	1 if the value is too low to use as a 100% reference voltage, currently set at < 0.020 V
	2 if the value is too high to use as a 100% reference voltage, currently set at > 3.200 V
	Persist through power-cycle:No
	Example return (1.421… volt 100% or “full scale”):
	`1421045`

`setautaveraging`	Syntax: `setautaveraging` What it does: This command sets the amount of averaging performed by the device to be set automatically based on the transimpedance amplifier voltage as follows: For firmware levels < 2.0.0.0 Voltage > 30mv, averaging equivalent of `setlowaveraging`

	3mv < Voltage < 30mv, averaging equivalent to `setmedaveraging` Voltage < 3mv, averaging equivalent to `sethiaveraging` For firmware levels > 2.0.0.0 Current > 10uA, averaging equivalent of `setlowaveraging` 0.32uA < Current < 10uA, averaging equivalent to `setmedaveraging` Current < 0.32uA, averaging equivalent to `sethiaveraging` This is the default setting for averaging upon power-up. Normal return value(s): 0 on success Error return value(s): N/A Persist through power-cycle:No

`setcurrentloop` (Gen2 only, firmware 2.0.0.5 and later)	Syntax:
	`setcurrentloop log`
	`setcurrentloop midpoint`
	`setcurrentloop [min picoamps] [max picoamps]`
	`setcurrentloop [0-24]`
	What it does:
	This command controls the 4-20mA current loop output of devices that support such a current loop. NOTE: for the current loop to behave properly, the current loop needs to be powered before the device.
	
	`setcurrentloop log` sets the device to output a logarithmic scale current that is in relation to the current sensed by the detector. The transfer function is:
	
	4-20mA current = (LOG10(detector current)+8)*3+5
	Detector Current = 10^(([4-20mA Current]-5)/3-8)
	
	If the detector current is below 10 nA, the 4-20mA current is set to 4mA. As a result, measurements fluctuating around 10nA will result in the 4-20mA output bouncing between 4mA and 5mA. If a calibration factor is set, the 4-20mA output will have a similar relationship to irradiance, but multiplied by the calibration factor.
	
	`setcurrentloop midpoint` sets the device to output a linear scale where:
	
	4mA = 0 detector current
	12mA = the detector current when this command was set
	20mA = double the detector current when this command was set
	
	`setcurrentloop [min picoamps] [max picoamps]`sets the device to output a linear scale where:
	
	4mA = min picoamps
	20mA = max picoamps
	
	`setcurrentloop [0-24]` sets the device to manually output the current indicated, in milliamps.
	
	Normal return value(s):
	0 on success
	Error return value(s):
	-500 if command not supported, i.e. on Gen1 devices
	-501 if missing fields
	-502 if there is a bad current loop value (applies to setcurrentloop [0-24])
	Persist through power-cycle:
	Yes for `setcurrentloop log`, `setcurrentloop midpoint`, and `setcurrentloop [min picoamps] [max picoamps]`.
	No for `setcurrentloop [0-24]`, which is intended as more of a test function.

`setdatetime` (Gen2 only)	Syntax: `setdatetime 12/05/2013 19:02:05` What it does:

	This command sets the real time clock's date and time. It accepts either of the following formats: mm/dd/yyyy hh:mm:ss mm/dd/yy hh:mm:ss The device is designed such that this date/time setting is UTC/GMT time. The device stores all date/timestamps in Epoch time format, which is later read out and converted to local time by an application. This is the case with the Data Logger software. Normal return value(s): 0 on success Error return value(s): -500 if missing fields -501 if parameters are out of range Persist through power-cycle:Yes, assuming coin/cell battery is in place with adequate charge
`sethiaveraging`	Syntax: `sethiaveraging` What it does: This command sets the averaging to be performed by the device to high, resulting in approximately 1 sample every 2 seconds. This mode provides the highest amount of over- sampling, or noise reduction. This mode is typically used for either detecting very low light levels or for monitoring noisy light sources. Normal return value(s): 0 on success Error return value(s): N/A Persist through power-cycle:No
`setlowaveraging`	Syntax: `setlowaveraging` What it does: This command sets the averaging to be performed by the device to low, resulting in approximately 5 samples per second. Normal return value(s): 0 on success Error return value(s): N/A Persist through power-cycle:No
`setmedaveraging`	Syntax: `setmedaveraging` What it does: This command sets the averaging to be performed by the device to medium, resulting in approximately 2 samples per second. Normal return value(s): 0 on success Error return value(s): N/A Persist through power-cycle:No

`setuserdark`	Syntax:
	`setuserdark`
	What it does:
	This command captures the detector voltage signal, at all automatic-gain-control stages, and stores the values in device flash memory. When used in conjunction with `useuserdark`, all subsequent readings will have this value removed automatically. Starting in firmware version 2.0.0.5, if the mode is already set to `useuserdark` then the new values will automatically be applied.
	Normal return value(s):
	User dark value, in microvolts, for all gain stages (separated by a space)
	Error return value(s):
	-500 if error setting dark value
	Persist through power-cycle:Yes
	Example return:
	`11289 10728`

`startlogdata`	Syntax:

	`startlogdata` [variable bitmask] [logging period / 10] [seconds since 1970]
	What it does:
	This command defines the logging parameters, and immediately starts logging. This command cannot be run when either a logging session is already active or a session has been stopped (with `stoplogdata`), but the log data has not yet been erased with `eraselogdata`. The command takes three parameters, with a space between parameters, as follows:
	[variable bit mask]
	1=Optical Density (x100)
	2=Percent Transmission (x10)
	4=Detector Current (picoamps)
	8=Detector Voltage (microvolts)
	16=Device board temperature (degrees F)
	32=Calibrated Irradiance (see `getirradiance`)
	128=Use Real Time Timestamps (as opposed to relative time stamps, Gen2 only)
	[logging period / 10] – *for firmware versions 2.0.0.1 and earlier*
	A 1, for example, would indicate a 10 second delay between logging; a value of 360 would indicate a 3600 second delay, or 1 hour, between log entries. The maximum value is 8640, which is equivalent to 86400 seconds or 1 day. Any values above this will results in a 1 day logging period.
	[logging period] – *for firmware versions 2.0.0.2 and later*
	A 1, for example, would indicate a 1 second delay between logging; a value of 3600 would indicate a 3600 second delay, or 1 hour, between log entries. The maximum value is 86400, which is equivalent to 86400 seconds or 1 day. Any values above this will results in a 1 day logging period.
	[seconds since 1970]
	This is "Unix epoch time", with most application coding environments having a mechanism to convert a date-time structure to epoch time. This is ignored, and should be set to 0, when using real-time timestamps.
	Normal return value(s):
	0 on success
	Error return value(s):
	-500 missing parameters
	-501 session already started. Must use `stoplogdata` and `eraselogdata` to start new.
	-502 errors with the variable bit mask
	Persist through power-cycle: Yes.
	Common usage is to use startlogdata, disconnect the device from the computer, connect the device to a battery or AC-power in the lab or field, and the device will continue logging on power up using time-stamps that are relative to the start time. In this scenario care must be taken to ensure any power down time is negligible to the desired time-stamp accuracy. Gen2 devices have the option of using a battery-back real-time clock (see 128 bitmask value above).
	Example command (to log detector current and device board temperature, every minute (Firmware 2.0.0.2), starting at 09 Sep 2013 14:50:00 GMT. Note bitmask of 20 is 4/current + 16/temperature):
	`startlogdata 20 60 1378738200`
	Example command (same as above, but adding 128 to the bit mask for use of a real time clock on a Gen2 device):
	`startlogdata 148 6 0`

`stoplogdata`	Syntax: `stoplogdata` What it does: This command stops a log session that was started with `startlogdata`. Note that, because date-time-stamping is relative to the start time set with `startlogdata`, logging cannot be restarted after a `stoplogdata`. Instead, log data must first be erased using `eraselogdata`. Normal return value(s): 0 on success Error return value(s):

	-500 if no active logging session to stop
`usecalfactor`	Syntax: `usecalfactor` [calibration number, 0-20] What it does: This command selects a particular calibration factor (see `setcalfactor`), which in turn will define the detector current-to-irradiance multiplier as well as the detector saturation current. A calibration number of 0 is a special case. Using 0 results in no longer using any calibration factors. When this is done, `getirradiance` will only return a value if `setsimpleirrcal` or `setirrdatapoint` has been used. Normal return value(s): 0 on success Error return value(s): -500 if missing fields -501 if factor number is outside the 0-20 range -502 if factor not defined Persist through power-cycle: Yes (firmware Version 2.0.0.0 and later), No otherwise Example command: `usecalfactor 5`
`usefactorydark`	Syntax: `usefactorydark` What it does: This command will cause all subsequent readings to remove the factory dark value before presenting any voltage, current, etc. readings. This is the default dark setting upon power up. Normal return value(s): 0 on success Error return value(s): -500 if there has been no factory dark value set Persist through power-cycle:No
`usefeedbackres (Gen2 only)`	Syntax: `usefeedbackres [resistor selection 0-4]` What it does: For Gen2 devices, which have 3 available feedback resistors, this command selects the resistor to be used as follows: 0: have the device automatically select a resistor based on light-level 1: use feedback resistor #1, usually the lowest value resistor 2: use feedback resistor #2, usually the middle value resistor 3: use feedback resistor #3, usually the highest value resistor Normal return value(s): 0 on success Error return value(s): -500 if missing parameters -501 if the device is not a Gen2 device -502 it the resistor selection value out of range -503 resistor selected, but error saving the change to flash memory Persist through power-cycle:Yes
`usenodark`	Syntax: `usenodark` What it does: This command will eliminate any dark current consideration. Normal return value(s): 0 on success Error return value(s): N/A Persist through power-cycle:No
`useuserdark`	Syntax: `useuserdark` What it does: This command will cause all subsequent readings to remove the user dark value (see `setuserdark`) before presenting any voltage, current, etc. readings.

	Normal return value(s): 0 on success Error return value(s): -500 if there has been no user dark value set Persist through power-cycle:No
*erasecalfactor	Syntax: `erasecalfactor` [calibration number, 1-20] What it does: This command erases the calibration data associated with the calibration factor. This includes the calibration factor description, the current-to-irradiance multiplier, and the saturation current. Normal return value(s): 0 on success Error return value(s): -500 if missing fields -501 if bad factor number -502 if error erasing flash Persist through power-cycle:Yes Example command: `erasecalfactor 5`
*eraseirrdata	Syntax: `eraseirrdata` What it does: This command erase any flash memory associated with the tabulated current-to-irradiance curve established with either `setsimpleirrcal` or multiple calls to `setirrdatapoint`. Normal return value(s): 0 on success Error return value(s): N/A Persist through power-cycle:Yes

*getcalfactor	Syntax:
	`getcalfactor` [optional, calibration factor, 1-20]
	What it does:
	Without the optional parameter, this command returns the calibration factor currently in use. With the optional parameter, this command returns the calibration factor details for a particular calibration factor. This includes the following information, separated by space: calibration factor description, current-to-irradiance multiplier(x1000), and saturation current in microamps. A typical use case is to use the command without the optional parameter to determine which calibration factor is in use, followed by issuing the command with the optional calibration factor to determine the details of the calibration factor definition.
	Normal return value(s):
	Without the optional command line parameter:
	
	0 if no calibration factor in use
	1-20 indicating which calibration factor is in use
	
	With the optional command line parameter:
	
	[calibration factor description] [current-to-irradiance multiplier(x1000)] [saturation current]
	Error return value(s):
	-501 if calibration factor is out of the 1-20 range
	-502 if the calibration factor is not defined
	Example command:
	`getcalfactor 1`
	Example return:
	`calfact1 500 50`

*getclockfreq	Syntax: `getclockfreq`

	What it does: This command returns the clock frequency, in MHz, as defined by `setclockfreq`. If `setclockfreq` has not yet been called, the default clock frequency is returned. See `setclockfreq` for more detail. Normal return value(s): Clock frequency, in MHz, times 100 Error return value(s): N/A Example return (for 80.41 MHz frequency): `8041`

`*getfeedbackres`	Syntax:
	`getfeedbackres`
	What it does:
	This command returns the value of the feedback resistor, in kilo-Ohms x 10.
	Normal return value(s):
	Transimpedance amplifier feedback resistance in kOhms x 10
	Error return value(s):
	N/A
	Example return (for a 3K Ohm feedback resistor):
	`30`

`*getvAGC3`	Syntax:
	`getvAGC3 (was getvx101 prior to FW 2.0.0.0)`
	What it does:
	This command returns the voltage at the 3rd automatic gain controller (AGC) stage. For "Gen1" products this is a x101 stage. For "Gen2" products, this is a x131 stage.
	Normal return value(s):
	AGC3 voltage in microvolts
	Error return value(s):
	N/A
	Example return (1.034.. volts):
	`1034054`

`*getvx1`	Syntax:
	`getvx1`
	What it does:
	This command returns the voltage at the x1 automatic-gain-control stage, in microvolts.
	Normal return value(s):
	x1 voltage in microvolts
	Error return value(s):
	N/A
	Example return (1.543.. volts):
	`1543087`

`*getvx17`
`(Gen2 only)`
	Syntax:
	`getvx17`
	What it does:
	This command returns the voltage at the x17 automatic-gain-control stage, in microvolts. It only applies to Gen2 devices that have this gain stage
	Normal return value(s):
	x17 voltage in microvolts
	Error return value(s):
	-500 if command not supported, i.e. on Gen1 devices
	Example return (0.782.. volts):
	`782512`

`*setcalfactor`	Syntax: `setcalfactor` [calibration factor, 1-20] [desc.] [multiplier x 1000] [saturation current uA] What it does: This command defines the particular calibration factor details. Details include the calibration factor description (up to 100 characters), the current-to-irradiance multiplier(x1000), and saturation current in microamps. If the multiplier value is 0xFFFF (65535) the calibration table as established by

	`setirrdatapoint` will be used to convert detector current to irradiance. Normal return value(s): 0 on success Error return value(s): -500 if missing fields -501 if calibration factor is out of the 1-20 range -502 if multiplier < 0 -503 error saving to flash Persist through power-cycle:Yes Example command ("calfact1" description, 0.500 current-to-irradiance factor, 50 uA saturation current): `setcalfactor 1 calfact1 500 50`
`*setclockfreq`	Syntax: `setclockfreq` What it does: This command starts a procedure that records an accurate measurement of the device microcontroller clock frequency. While the clock frequency is nominally 80.00 Mhz, in reality it is slightly different from this value. Because this clock frequency will be used for time-stamping, it is recommended to use `setclockfreq` to calibrate the time-stamping routines. This command cannot be run during an active logging session because it must take control of the timer interrupt, which is active during logging. Procedure: This is the only device command that is more involved than a simple command/response sequence. If the device is being run in the `echoon` mode, from a terminal server or CLI, the device will prompt the user through the proper procedure. If the command is being driven by an application, in `echooff` mode, the procedure is as follows (example source code is available from ILT if needed): 1. send the `setclockfrreq` command 2. send the single character "A" 3. wait exactly 60 seconds 4. send the single character "B" Normal return value(s): 0 on success Error return value(s): -500 if a logging session is active -501 if there is an error with the procedure -502 if the clock frequency measured is off by more than 1MHz from the 80MHz nominal Persist through power-cycle:Yes
`*setsamplecount (firmware 2.0.0.6 and later)`	Syntax: `setsamplecount` [1 – 200] What it does: This command sets the number of samples taken by the Analog-to-Digital converter when reading the voltage from the transimpedance amplifier. This is the immiedate averaging done on the input signal, before additional averaging is done by the `set*averaging` commands. Typically a value of 200 is used (the default) for standard operation, or a value of 1 is used for high-speed sampling (up to 100 samples/sec). WARNING: modifying this value will impact the communication between the ILT device and any device talking to it over the USB port. Talk to the manufacturer for details. Normal return value(s): 0 on success Error return value(s): -500 if missing fields -501 if value is out of range Persist through power-cycle:Yes Example command (for sampling at 100/sec): `setsamplecount 1`

	Example command (for default sampling rate): `setsamplecount 200`

Theory of operation:

ILT1000 and ILT5000 "USB Serial Port" Programming / Theory of Operation

Below is the Theory of Operation for interfacing with the ILT1000 or ILT5000 over a serial port. "Device" below refers to either the ILT1000 or ILT5000, as the Theory of Operation is the same.

- The device responds to a number of commands (together known as the command-line API) delivered via the USB port.
    - Contact your ILT representative for a complete API guide.
- The USB port is configured on the device as a USB Serial Port. As a result, interfacing can be done with any standard terminal program (hyperterminal, putty, MAC terminal window, etc).
- The device behaves as follows and in sequence:
    a. Perform analog/digital conversion and mathematical functions as part of optical level detection
        i. Buffer up to 4 characters of any incoming commands while performing the processing above
    b. Check for any characters in the buffer referenced above to indicate an incoming command and process any commands as required. Important programming notes:
        i. It can take up to 25ms for the system to complete complex analog/digital tasks and process the remainder of the command that was not buffered.
        ii. The device determines the completion of the command by detecting a "\r" character, i.e. ASCII code 13 decimal.
    c. Start all over from the top at (a)….
- As a result of the above device behavior, the recommended method to program each device is to:
    - The device operated at 115200 baud, 8 bits, 1 stop bit, no parity bits, no flow control.
    - Always append commands with \r to indicate the command completion.
    - Send the first character of the command, for example the "g" in "getcurrent\r".
    - Pause 50ms (this can be lowered for high speed access, but start with 50ms)
    - Send the remainder of the command, i.e. "etcurrent\r". There are no pauses required between characters as the device is rapidly draining the incoming serial port at this point.
    - Immediately start sensing the response from the device.
        - The device will always send a response to acknowledge the status of the command completion as well as to return values for "get" commands.
        - The response will always be terminated with a "\r\n" (13 decimal, 10 decimal) sequence. This can be used to sense the end of the response and start sending the next command.
            - Regarding time-outs for command responses, "get" commands will typically respond within 100ms. "set" commands that store their configuration in flash memory can take up to 5 seconds to respond. Other special commands like "setfactorydark", "setuserdark", and "captureflash" can take longer to respond.
- Each ILT1000/ILT5000 device is single threaded, meaning that commands and responses need to be processed in sequence. A 2nd command cannot, for example, be

initiated before the 1st command's response is fully processed. As a result, if a multi-threaded application is accessing the device, a programmatic lock must be placed around device command/response sequences to make sure multiple threads do not attempt to access the device at the same time.

- If multiple devices are being monitored, a single lock can be used for access to all devices. This is simpler from a coding perspective, but not as efficient as it does not allow multiple devices to operate in parallel. For the best performance it is recommended that a per-device lock be established. This allows all devices to be accessed in parallel.
- As a further performance benefit when monitoring multiple devices, the delay after the first character can be performed in parallel across all devices. For example, if sending "getcurrent\r" to 5 devices, one would:
  - Send "g" to all 5 devices
  - Wait 50ms
  - Send "etcurrent\r" to all devices
  - Process the reply from all devices

Labview code, Sample program:

You can download a copy of our labview vi from the ILT software page on the ILT website:
http://www.intl-lighttech.com/support/software

To send command to the ILT1000, you need to send the first character, wait 50ms, then send the remaining characters. You can do this in a couple of ways:
1. Send the characters 1 by 1 and leave a 50ms delay between each. This is how the example Labview code works to simplify the block diagram.
2. Send the characters 1 by 1, leave a 50ms delay after the first characters, then send the others without delay.

Others notes:
1. \r needs to be the only trailing characters to indicate the end of the command
2. On receipt, read until you get a \n

For this device, the only setup should be for serial port: 115200,N,8,1,no HW or SW handshakes.