

DynaVision®



MULTIPOINT SCANNER SYSTEM NPH-66 API INTERFACE

USER MANUAL

by

LMI Technologies Inc.

Revision: 5.1.4.0.
January 2007



PROPRIETARY

This document, submitted in confidence, contains proprietary information which shall not be reproduced or transferred to other documents or disclosed to others or used for manufacturing or any other purpose without prior written permission of LMI Technologies Inc.

LMI Technologies Inc.

1673 Cliveden Ave.
Delta, BC V3M 6V5
Canada
Telephone: 604-636-1011
Fax: 604-516-8368
www.sensorsthatsee.com

Trademarks and Restrictions

DynaVision® is a registered trademark of LMI Technologies Inc.

This product is designated for use solely as a component and as such it does not comply with the standards relating to laser products specified in U.S. FDA CFR Title 21 Part 1040.

Windows 3.1®, Windows 95®, Windows 98®, and Windows NT® are registered trademarks of Microsoft Corporation.

No part of this publication may be copied, photocopied, reproduced, transmitted, transcribed, or reduced to any electronic medium or machine readable form without prior written consent of LMI Technologies Inc.

Printed in Canada.

Table of Contents

1. Overview.....	4
1.1. B8A Multipoint Sensor Board Scanning System	4
1.2. B16 Multipoint Sensor Log Scanning System	4
1.3. NetPowerHub Enclosure	5
1.4. Sensors	6
2. Applications	7
2.1. Board Scanning System	7
2.2. Log Scanning System.....	10
3. Net Power Hub	14
3.1. Scanner Parameters Structure	14
3.2. NPH Operation Modes.....	15
3.3. NPH Inputs/Outputs.....	19
3.4. Scanner System Power Specifications.....	22
4. NPH API Functions	24
4.1. NbLib API Functions	24
4.2. NbLib API Errors.....	44
5. Scanner System.....	46
5.1. Scanner System Connection	46
5.2. Warnings and system design considerations	47
5.3. Sensor Firmware Upload Procedure	48
5.4. NPH IP Address Setup	50
5.5. NPH Firmware Update.....	53
5.6. NbTest Diagnostics Program.....	58
6. Getting help.....	66

1. Overview

1.1. B8A Multipoint Sensor Board Scanning System

B8A sensor based scanning systems are primarily designed to provide true shape, 3D measurements of lumber to enable computerized optimization of lumber processing centers such as canters, edgers, trimmers and sorters. Raw boards (flitches) are fed under the scanner using cross-transfer mechanism such as chains equipped with lugs. Each set of lugs is "pushing" a raw board (flitch) through the scanner. Travel distance is usually measured using opto-mechanical encoder attached to the motor. While the flitch is traveling under the scanner at each encoder pulse the scanner takes range measurement for each measurement point, in B8A based systems at resolution of N" (3" for B8A and 1" for M24B) along the length of the target. These measurements are stored in the internal board data buffers and made available to the user (client computer) once the target board completely passes through.



1.2. B16 Multipoint Sensor Log Scanning System

B16 sensor based log scanning systems are primarily designed to provide true shape, 3D measurements of logs to assist computerized optimization of log processing. Logs are transferred under the scanner using lineal-transfer mechanism such as sharp chains or belts through the scanner zone. Travel distance is usually measured using opto-

mechanical encoder attached to the motor. While the log is traveling under the scanner at each encoder pulse the scanner takes range measurement for each measurement point, in B16 based systems at resolution of 25.0 mm (aprox. 1") around the log circumference. These measurements (scan frames) are stored in the NPH internal data buffers and transmitted to the client CPU.

Embedded NPH software provides board/target detection functionality and the Ethernet communication link with the AIP library (*nbllib.dll*) residing on a client computer. Board detection algorithm is setup by user modifying detection parameters at the time of NPH initialization. LMI provides the OEM software support for Win32 OS platform.

Windows NT / 2000 / XP OS API software is distributed as four files:

nbllib.h header file containing function prototypes declarations, structure type-definitions and useful constants. This file is required to build any application that uses the library.

nbllib.lib is an import library for nbllib.dll that must be included in the application development projects.

nbllib.dll is a run-time dynamic-link library containing the library's executable code. This file should be installed either in the Windows system directory (typically c:\winnt\system32 under Windows NT) or specific application directory.

1.3. NetPowerHub Enclosure

The NetPowerHub concentrator distributes individually fused DC power to each installed sensor as well as provides both proprietary high speed and RS-485 serial communication. The NetPowerHub must be mounted inside an industrial NEMA enclosure at the scanner frame to protect the electronics and is mounted on the back panel of that enclosure.



1.4. Sensors

1.4.1. B800/B8A Multipoint Range Measurement Sensor

Each B8A sensor projects 8 laser beams onto a target surface at 3" (76.2 mm) spacing to cover 2" section of a target (board) length. A laser beam incidence on the target (laser spot) is detected by two integrated CCD line cameras and the target range is calculated using the triangulation method. Because of the (patent pending) binocular arrangement of cameras, ranges are calculated for all lasers at the actual scan rate of the CCD, without slowing the sensor or missorting spots.

The B8A sensor uses only solid state components. The lasers used in the LMI DynaVision sensors emit visible red spectrum of 670 nm wavelength (class IIIA). Collimated laser's intensity is high enough to damage the eyes if viewed directly. During normal operation, the lasers do not pose any hazard to the mill personnel. The only situation in which the lasers could be viewed directly would be when personnel are working inside the throat of a powered scanner frame. Therefore it is mandatory to power down scanner sensors while working within the scope of the scanner.

1.4.2. B16 Multipoint Range Measurement Sensor

Each B16 sensor projects 16 laser beams onto a target surface at 25 mm (0.984") spacing. A laser beam incidence on the target (laser spot) is detected by an integrated CCD line camera and the target range is calculated using the triangulation method.

The B16 sensor uses only solid state components. The lasers used in the LMI DynaVision sensors emit visible red spectrum of 670 nm wavelength (class IIIA). Collimated laser's intensity is high enough to damage the eyes if viewed directly. During normal operation, the lasers do not pose any hazard to the mill personnel. The only situation in which the lasers could be viewed directly would be when personnel are working inside the throat of a powered scanner frame. Therefore it is mandatory to power down scanner sensors while working within the scope of the scanner.

2. Applications

2.1. Board Scanning System

2.1.1. Board Scanning Concept

Boards are transported in a transversal fashion through the scanner frame. The transport mechanism, such as chain is coupled with an opto-mechanical encoder device measuring travel distance in given encoder resolution. One encoder pulse corresponds to certain transport displacement. The scanner operates in several modes that are explained in detail under NPH Modes of Operation section.

2.1.2. Scanner System Components

Following scanner components are supplied by LMI Technologies:

- Range measurement sensor(s).
- NetPowerHub concentrator unit (NPH-66) which collects data from up to 24 sensors.
- Integrated data / power Cable for power and serial communications.

The OEM must supply the following components:

- Scanner frame to mount the sensors.
- Power supply to provide DC power to the sensors (24V power supply recommended, although the sensors accept 14.5 to 26 Volts). Each model sensor varies in the current rating, note the NPH uses 500mA itself, include this when calculating the maximum current rating of the power supply.
- Power supply cabinet to host the DC power supply and the NPH Concentrator.
- Differential quadrature opto-mechanical encoder coupled mechanically to the transfer mechanism to produce pulses in proportion to the chain travel.
- Encoder Cable to connect the encoder to the NPH DB-9 input connector.
- Client computer with a NIC Ethernet card to run data processing software such as optimizer and diagnostics software.
- UPS Uninterruptible power supply to provide 120 VAC to the client computer and the power supply cabinet which powers the NPH.
- System Calibration Bar. A 2" by 4" by "board length" profile bar to calibrate the sensor system.

2.1.3. Scanner System Installation

The procedure describes the step by step instructions to install the DynaVision scanner system.

Frame Consideration

Isolate the scanner from mill vibration, the frame should be mounted on an independent sub-base to the mill foundation. Shock mounts should be used to mount the scanner frame to the sub-base. The frame height should ensure the level of the transfer chain is correct according to drawings. The first sensor is located inside the even ending zero lumber line. This will locate the first laser scanning line 1"-3" in from the zero lumber.

Sensors are numbered from the zero lumber line. 0' top sensor, 2' top sensor ... N' top sensor 0' bottom sensor, 2' bottom sensor ... N' bottom sensor etc.

Power Supply Cabinet Consideration

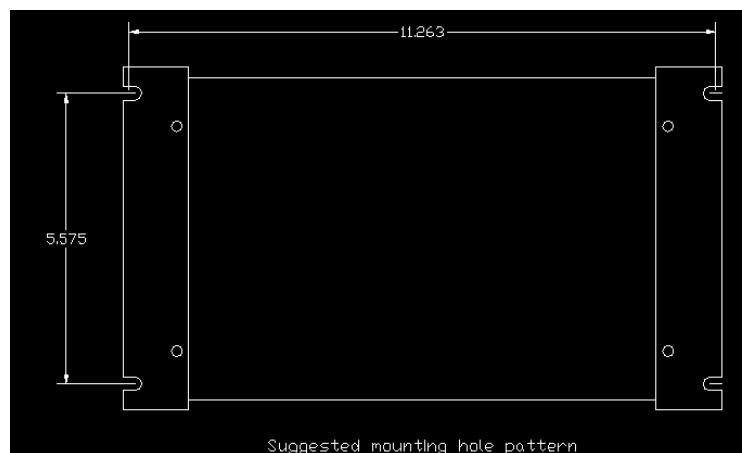
Mount the power supply cabinet on one end of the scan frame to allow proper access from the mill floor for installation and service. Position the cabinet on the end of the frame so there is proper clearance from the bottom access plate on the frame into the power supply cabinet to run the power cables. Ensure that any warning bulbs mounted on the scanner frame or power supply cabinet is not positioned where they are in direct view of any of the sensors cameras.

DC Power Check

Connect DC power to NPH Wiring concentrator (before connecting to sensors). Turn on the DC power and verify the DC power is correct at the DC power studs, and that the Concentrator red light goes on. The red light will not light if the power is reversed (or not connected, of course). NOTE: Both the NetPowerHub-66 and the sensors include reverse and over voltage protection. This is a severe condition, however, and should not be deliberately tested.

Install NPH and Integrated Power cabling

Fish the integrated power cabling through the hole in the scanner frame (adjacent to each sensor), and out of the access cutout up into the power supply cabinet. There is one power cable per sensor. Label each power cable as 0' Top, 0' Bottom, etc. so that you know where to wire them in the cabinet. Any excess cable length should be inside the frame tubing. Connect Integrated Power cable to the NetPowerHub-66. Position is not significant (any cable can go to any available connector). Suggested bolt size: #8 to #10



Installing sensors

The sensors are held in place at three points (mounting holes). Two of the points (connector side) are mounted onto studs to allow the easy installation of shims to aim the sensors in case the frame is not aligned correctly. The other point is mounted with a hex cap screw. The studs should be installed in the frame with Loctite and left to set to ensure studs are tight. Install all the baffle plates but do not tighten the mounting bolts yet (hand tight only so the plate sits flat on the scan frame. The flat baffle plates are

located on the top and angled baffle plates on the bottom. There may be two types of plates to handle the 1/2" side to side stagger between sensors. Ensure that the slots for the laser beams match the rows of lasers from each sensor. With the AC power OFF to the scanner frame cabinet, plug the integrated power cable into each sensor. Ensure that the cables are fully pushed into place and the hold down sleeve screwed on hand tight. Turn on the AC power to the scanner frame cabinet. After a delay the DC power will be supplied to the sensors. Observe the LCD display on the sensors. Each sensor will cycle through and display a checklist of internal DC Voltage values. The final display shows a bar graph of the range for each laser spot. The further away the object is detected the longer the bar if there is not display for any of the sensors check the power connection to that sensor. The next time the power is turned OFF then ON observe the voltage values displayed on that specific sensor for any indication of low voltage. If the external voltage is low to any of the sensors then it will not function properly and appear that it is not working. If low current is suspected then check that the power supply has sufficient capacity to supply the complete frame (# of sensors x 700 mA) or check that the current limit adjustment for the power supply is set correctly. **Warning! Ensure the power is off when installing or removing a power cable from a sensor.** Adjust the top and bottom sensors until the lasers from the top sensors pass through the holes in the bottom sensor laser windows, and vice versa. Adjust the baffle plates so the no lasers are hitting the baffle plates. Note that some secondary reflections from the laser windows will always hit the inside of the baffle plate, however, they are outside of the sensor's field of view and not a concern. The LCD display is also useful as a final check for aligning the baffle plates. After aligning with sensors and baffles visually, the LCD display should show that no spots are being detected. A spot being detected would appear as a longer line on one or more of the bars on the graph. If one or more lasers are being detected then check that the baffle plate is aligned or the specific hole is blocked or misaligned. If any of the lasers are detected on a chain or support structure then make note of which laser and which sensor this is occurring on, as the NPH software is used to disable the spot detection using the 'nbSetLaserStatus()' command. The beam should also be physically blocked with masking tape over the window to avoid any possible reflections. **The above mentioned installation procedure does not apply to the B16 sensor based scanning system since there is no LCD implemented on these sensors.**

Installing Photocells

Scanner system accepts up to 8 opto-coupled photocells inputs via DB-15 connector on the face of the NPH concentrator.

Installing the Encoder

The encoder should be installed to be driven from the scanner transfer chain. The encoder is wired to the computer cabinet with a shielded cable in separate conduit from the encoder to the NPH unit via DB9 connector.

System Check

Scanner system check is performed after the entire scanner is connected and cabled to the NetPowerHub concentrator and the client computer. This checkout makes extensive use of the NBTEST.EXE diagnostics software. After starting NBTEST.EXE on the client computer, press 'Connect' dialog button on the initial screen. Ensure that sensors are detected.

2.2. Log Scanning System

Following is the concepts description of a log scanning systems using multiple B16 sensors arranged in a circular fashion around the transport mechanism and the NPH-66 concentrator. However these modes of employment are not restricted to log scanning only. Board/cant scanning application may find these modes useful.

2.2.1. Log Scanning Concepts

Different log measurement applications require different ways of collecting real-time measurements to put together a digital log model for optimization. For logs transported in a longitudinal fashion through the scanner frame the scanner system typically requires 3 to 4 sensors concentrically arranged around the in feed with scanning plane perpendicular to the centroid of the log. The transport mechanism, such as sharp chain or a belt is coupled with an opto-mechanical encoder device measuring travel distance in given encoder resolution. One encoder pulse corresponds to certain transport displacement. The scanner operates in two modes, user triggered (manual) and automatic log detection mode.

For veneer peeling applications the log is positioned under the scanner and rotated 360 degrees to collect all the measurements. In this case the sensors are arranged in a continuous array parallel to the longitudinal axis of the log. Once the mechanism starts rotate the spindle, coupled with an encoder measurements are taken around the log circumference. It takes one full rotation to collect all the measurements for the whole log at given longitudinal resolution (spacing of sampling points, laser beams).

NPH concentrator can be dynamically setup for both methods.

2.2.2. Longitudinal Log Scanning Mode

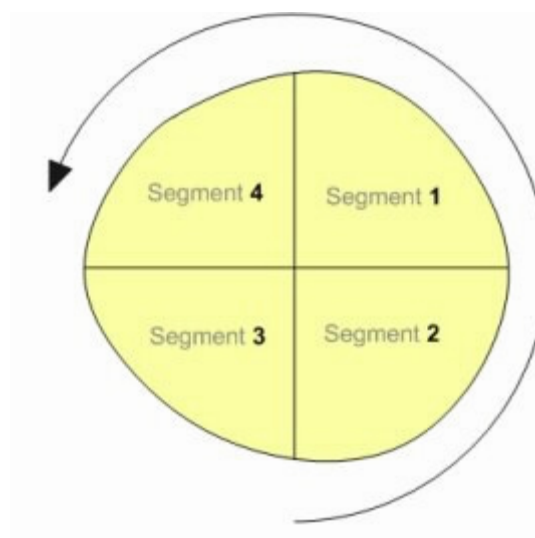
This scanning mode requires an external trigger, a photocell or a switch connected to one of the 8 available photocell inputs on the NPH concentrator unit. This trigger would be typically located just ahead of the scanning plane of the scanner. The trigger/photocell will change its state from **Light** to **Dark** once the log moving along the in feed reaches its scope. This causes the NPH unit to start buffering real-time range data from all the installed B16 sensors (called data frames) at each **forward** encoder pulse. NPH subsequently (typically within a millisecond) starts to transmit collected buffered data to the client computer over 100Mb Ethernet link using UDP streaming protocol. Each data frame has an encoder count and state of the photocell inputs associated with it. Meanwhile, at the client side application (nbllib.dll) collects these data frames and stores them in allocated buffers. To optimize the time data are being sent in predefined segments of N-encoder counts long (see **maxwidth** field of the scanner parameters structure). After receiving each data segment (N * data frame) the application, **nbllib.dll** will issue a *Board Ready Event* to the user application, typically the optimizer software. Upon detecting this event the user will copy available data segment to its own application data space and starts processing the data. Meanwhile another segment is being scanned and delivered to the client computer. This sequence repeats as long as there is a valid target under the scanner or a given photocell is in the **dark** state. The last segment is delivered once the log exits the scanner's scope, none of the B16 sensors detect the target anymore and the photocell used for the trigger is in the

light state. Thus complete log model consists of multiple segments of known length N , with last segment containing true end of the log being most likely less than N long. The actual end of log is located somewhere within the last segment and the user software must locate the last valid data and completely define the log model. The length of the log can also be determined using the photocell data to the encoder resolution using the encoder count associated with each scanned frame.

Next the scanner monitors the photocell state and waits for its next transition from light to dark (sometimes referred to as open and closed). While doing so even though the transport is moving and the encoder is issuing pulses the system is not busy and no data are being transmitted over the network. It is important that there is a sufficient gap between two consecutive logs since

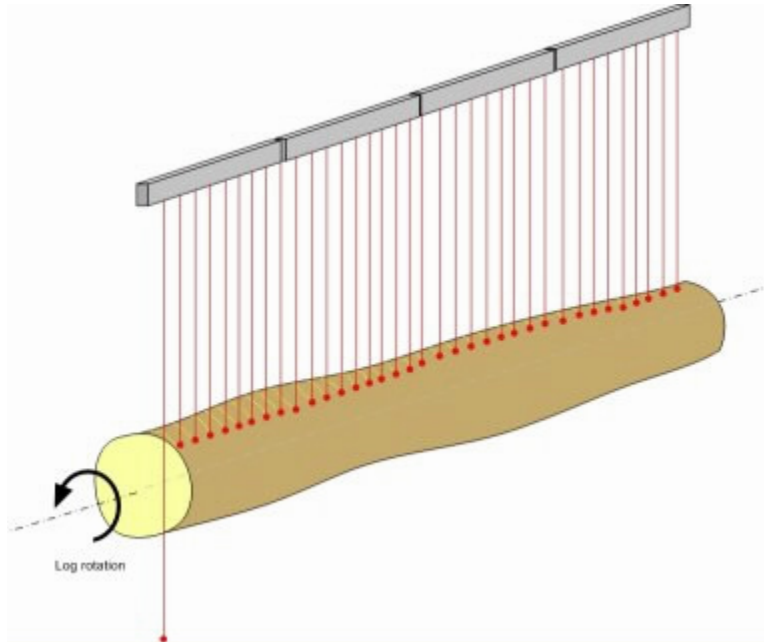
2.2.3. Rotational Log Scanning Mode

In this case a log positioned under the scanner on a spindle. Log is positioned within the scope of the scanner and an external trigger, similar to the longitudinal scanning is used to start taking log measurements. It takes one full rotation to collect all the samples to construct a digital model of the log. Let's say the encoder issues 360 pulses per rotation. This yields 360 measurements around circumference of the log, in other words user will acquire measurements at one degree resolution around the circumference and 1" resolution along the log. This mode of operation is suitable for applications such as veneer peeling. There is one more parameter associated with this mode. Rather than waiting for all the data to arrive and then start to process them, application can specify that system should segment the data. In the case above, 360 measurements per rotation we could specify that we want data in segments of 90 measurements. Thus the whole scan will be delivered in 4 segments, each 90 data frames wide. Upon receiving each segment the *Board Ready Event* is issued and it is the responsibility of the application to copy/move the buffered data to its application data space. Once the last segment is completed the system waits for the next trigger signal, regardless of the encoder activities or detection of any object under the scanner. Both methods have its advantages and it is up to the user to select the method most suitable for the application.



2.2.4. Continuous Scanning Mode

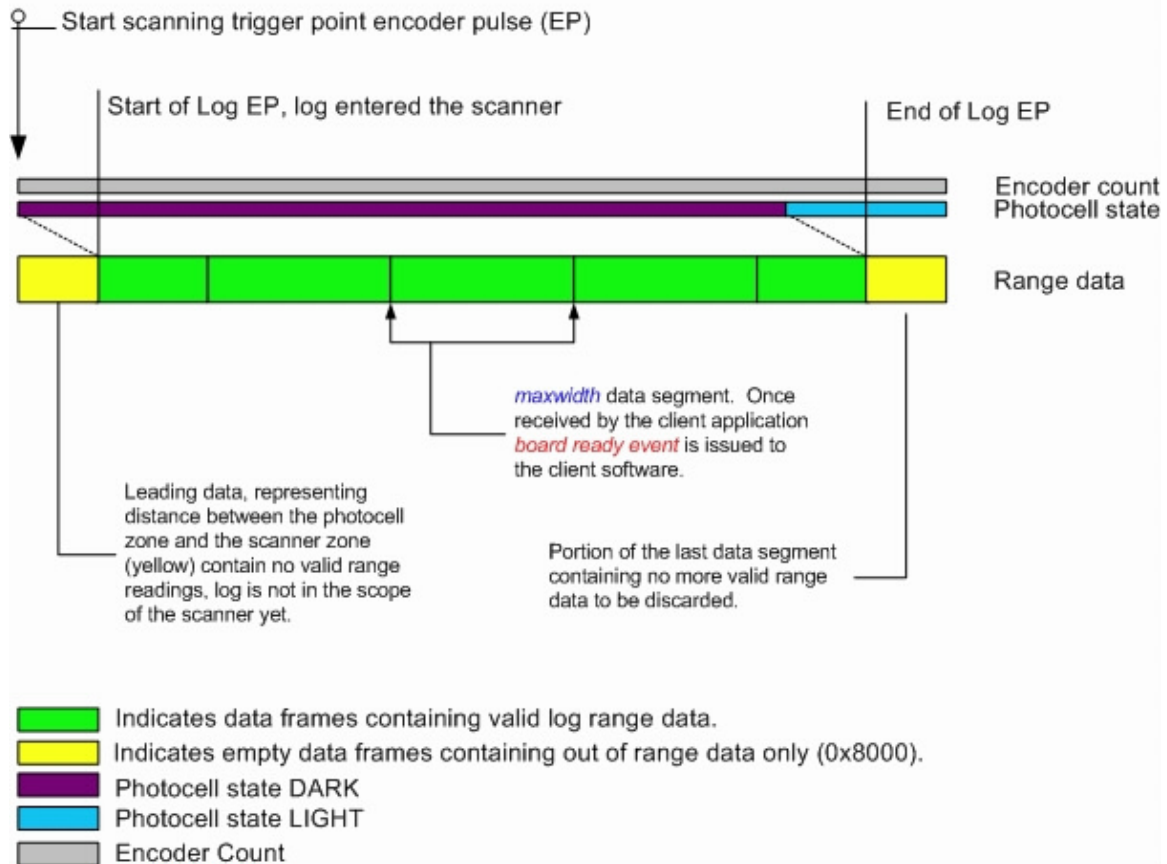
This mode of employment is quite rudimentary and relies completely on the application to analyze data and put together target/log models. Upon startup the scanner will start delivering data frames on each encoder pulse, forward or reverse. The application, *nbllib.dll* receives data in sections of *maxwidth* size and issues *Board Ready Event* after each of them. Upon each *Board Ready Event* the client application, optimizer transfers the section data into its own data space and executes data processing. Meanwhile another data section is being collected.



2.2.5. Log Length Information

On each specified photocell transition, light to dark and dark to light the NPH records current absolute encoder count. These counts are logged and immediately transmitted over the Ethernet connection using the UDP protocol to the client computer. Upon receiving this type of UDP packet the API will issue the `ASYNC_EVENT_LOG_PC_INFO` event. Once the event is issued user can import the photocell states information using the `nbReadLogLength( PCSTATEST *pcstates, int source)` API call. There are two benefits of sending this information to the user. One, user is able to calculate current length of the log accurately. Second, user has information on spacing of logs entering the scanner. This information may be useful to monitor efficiency of the whole log processing operation. The same information can be accessed by polling the NPH using the TCP/IP protocol. See `nbReadLogLength()` API call for more explanation.

Real Time Log Scanning Data in Time



3. Net Power Hub

3.1. Scanner Parameters Structure

Using LMI's API interface the user software is responsible for setting up the scanner parameters upon establishing network connection with the NPH unit. Following are the available scanner parameters (SCANNERPARAMS structure in the nplib.h file) that will affect the functionality of the scanner. For each scanner application, board or log scanner, only certain scanner parameters apply. Rest of them are not employed and have no impact on scanner functionality. Following table describes the use of these parameters:

lead_spots	This parameter is not being used in any log scanning modes. Minimum number of valid, not out of range (0x8000) readings to validate leading edge of a target, board. Once the scanner detects number of range readings greater or equal to this parameter it starts to count number of subsequent encoder pulses this condition holds (see lead_wait).
lead_wait	This parameter is not being used in any log scanning modes. User defined number of consecutive encoder pulses that the need to elapse while the system detects lead_spots of valid range readings to validate the target/log (see lead_spots).
trail_spots	This parameter is not being used in any log scanning modes. Similar to the leading edge this is maximum number of valid range readings scanner needs to report in order to keep collecting data. Once less that trail_spots are detected the scanner starts counting consecutive number of encoder pulses this condition holds. Once the count reaches trail_spots value the end of log state is issued.
trail_wait	This parameter is not being used in any log scanning modes. User defined number of encoder pulses that the end of log condition (see trail_spots) lasts.
lead_history	Applies to board scanners only. This parameter specifies how many data frames prior to detection of leading edge to include in a scan buffers. We recommend this to be about 5 encoder pulses. You should set it so that you have a buffer of out of range data before the scan data starts. Example <i>lead_wait</i> = 3, <i>history</i> = 5, <i>maxwidth</i> = 500 the total size of collected data frames will be 508.
trail_history	Applies to board scanners only. This parameter specifies how many data frames after detection of trailing edge to include in scan buffers. We recommend this to be about 5 encoder pulses. You should set it so that you have a buffer of out of range data after the scan data ends. Example <i>lead_wait</i> = 3, <i>history</i> = 5, <i>maxwidth</i> = 500 the total size of collected data frames will be 508.
maxwidth	In case of log scanning specifies maximum data segment length in encoder pulses (see Fig.1). In case of board scanning this parameter represents maximum expected board width in encoder pulses.
maxreverse	This parameter allows operator to reverse the transport for given number of encoder pulses without the need to rescan the whole piece. This value depends on log stability on the transport mechanism.

mode	Indicates a scanner mode. 0 - Automatic Board Detection Mode 1 - Longitudinal Log Scanning Mode 2 - Rotational Log Scanning Mode 3 - Continuous Log Scanning Mode 4 - Software Triggered Scanning Mode
pcindex_trg	Specifies external trigger photocell input index (0..7).
escale	Encoder scaling factor, i.e. value of 4 means that the data will be stored at the resolution of 4 encoder pulses (Every 4 th pulse will be stored).
pcindex_len	Specifies photocell input index (0..7) used for length measurement (must be different from the <i>triggerindex</i>).
segments	Specifies number of consecutive segments of maxwidth data frames, encoder pulses to deliver upon each trigger event. Data frames are collected regardless of content. System defaults to 1.
pcindex_stopscan	Specifies photocell input index (0..7) used for manually stopping scan (must be different from the <i>triggerindex</i> & <i>lengthindex</i> or it will be disabled). This is only for use in mode 1.
sample_freq	Specifies frequency at which the NPH will poll the sensors for data. 0 = 1000Hz 1 = 1500Hz 2 = 2000Hz The default value is 1000Hz. *This value is recommended to be set to 1000Hz, but if faster scanning is required it should be set according to the rate at which you will run your scanner system. 2000Hz is not possible for a system that includes B8, or M24B sensors as their max scan frequency should be set to 1000Hz.

3.2. NPH Operation Modes

Using LMI's API interface the user software is responsible for setting up the scanner parameters upon establishing network connection with the NPH unit. Following are the available scanner parameters (SCANNERPARAMS structure in the nblib.h file) that will affect the functionality of the scanner. For each scanner application, board or log scanner, only certain scanner parameters apply. Rest of them are not employed and have no impact on scanner functionality. Following describes different modes of operation and use of the scanner parameters:

3.2.1. Board Detection Logic (BDL) Mode (***scannerparams.mode = 0***)

This mode uses NPH capability to detect the leading edge of a board/flitch entering the scanner based on user defined parameters *_wait* sent to the NPH unit at the initialization time (*nbOpenScanner()*). Once the scanner detects *lead_spots* number (total per system) of valid range readings for duration of *lead_wait* number of encoder pulses it starts buffering and transmitting scanner data frames at each encoder pulse until the

scanner detects less than *trail_spots* number of valid (not 0x8000) range readings for more than *trail_wait* number of encoder pulses. Application running on the client computer receives all the data frames over the Ethernet network connection and once all the data frames are collected it will notify the user that all the data are available issuing a Board Ready event to the user software. At this time user will call *nbGetScannedObject()* API call to copy available board data to its own data space and can start processing them. Meanwhile a new board may already be being scanned. Example:

```
scannerparams.lead_spots      = 5
scannerparams.lead_wait      = 3
scannerparams.trail_spots     = 5
scannerparams.trail_wait     = 3
scannerparams.history         = 5
scannerparams.maxwidth       = 500
scannerparams.maxreverse     = 10
scannerparams.mode           = 0
scannerparams.pcindex_trg    = 0
scannerparams.pcindex_len    = 1
scannerparams.escale         = 1
scannerparams.segments       = 1
scannerparams.trail_holdscan = 5
scannerparams.pcindex_stopscan = 2
```

**scannerparams_pcindex_len* and *scannerparams_pcindex_trg* must be exclusive, even if not used!

In the above example, once the scanner registers 5 valid range readings in total (cumulative from all of the installed sensors) for 3 encoder pulses in a row, NPH will start transmit scan frames from that moment on. Let's say that the encoder counter value was 1000 (CEC) at that moment. The very first sent scan frame will be the one that was acquired at the time the encoder count was **1000(CEC) - 3(lead_wait) - 5(history)**. So the encoder count field associated with the first scan frame should be **902**. All of the consecutive scan frames collected from that moment on each encoder pulse will be sent to the client computer until either number of scan frames sent is equal *maxwidth*, in our case **500**, or the scanned object disappears from the view/scope of the scanner and thus satisfy trailing edge detection logic: there is less than 5(*trail_spots*) valid range readings returned by all sensors in total for more than 3(*trail_wait*) consecutive encoder pulses. Once either of these conditions are met the Board_Ready event is issued to user application. Should the first condition happen and the rest of the board is still in scanner's range, scanner will immediately initiate a second board acquisition. This is obviously not desirable, therefore *maxwidth* parameter should represent maximum width of a board (usually one lug space) expected. The only way for this to happen then is if a board is skewed over more than one lug space. In such a case it is recommended to re-scan the board again if possible or take precautions to prevent skewing of boards.

3.2.2. Photocell Triggered Mode (PCTM) I (*scannerparams.mode = 1*)

This scanning mode requires an external trigger, a photocell or a switch connected to one of the 8 available photocell inputs on the NPH concentrator unit. This trigger would be typically located just ahead of the scanning plane of the scanner. The trigger/photocell will change its state from **Light** to **Dark** once the target object moving

along the in feed reaches its scope. This causes the NPH unit to start buffering real-time range data from all the installed sensors (called scan frames) at each **forward** encoder pulse. NPH subsequently (typically within a millisecond) starts to transmit collected buffered data to a client computer over 100Mb Ethernet link using UDP streaming protocol. Each scan frame (one UDP packet) has an encoder count and state of the photocell inputs associated with it. Meanwhile, at the client side application (nbllib.dll) collects these data frames and stores them in allocated buffers. To optimize the time data are being sent in predefined segments of N-encoder counts long (see *maxwidth* < field of the scanner parameters structure). After receiving each data segment (N * data frame) the application, **nbllib.dll** will issue a *Board Ready Event* to the user application, typically the optimizer software. Upon detecting this event the user will copy available data segment to its own application data space using the nbGetScannedObject() API call and will start processing the data. Meanwhile another segment of scan frames is being scanned and delivered to the client computer. This sequence repeats as long as there is a valid target under the scanner or a given photocell is in the *dark* state. The last segment is delivered once the log exits the scanner's scope, none of the sensors detect the target object anymore and the photocell used for the trigger is in the *light* state. This last segment will consist of less than *maxwidth* number of scan frames. Thus complete scanned object model consists of multiple segments of known length = *maxwidth*, with last segment containing true end/trailing edge of the object being most likely less than *maxwidth* scan frames in size. Next the scanner monitors given photocell (defined by user on scanner initialization) state and waits for its next transition from light to dark (sometimes referred to as open and closed). While doing so even though the transport is moving and the encoder is issuing pulses the system is not busy and no data are being transmitted over the network. It is important that there is a sufficient gap between two consecutive objects to satisfy trailing edge detection constraints.

Example: User will request set of 5 segments of 50 scan frames each. User parameters will be set as follow:

```
scannerparams.lead_spots      = na
scannerparams.lead_wait      = na
scannerparams.trail_spots     = 5
scannerparams.trail_wait     = 3
scannerparams.history        = na
scannerparams.maxwidth       = 50
scannerparams.maxreverse     = 10
scannerparams.mode           = 1
scannerparams.pcindex_trg    = 0
scannerparams.pcindex_len    = 1
scannerparams.yscale         = 1
scannerparams.segments       = 1
scannerparams.trail_holdscan = 5
scannerparams.pcindex_stopscan = 2
```

*scannerparams_pcindex_len and scannerparams.pcindex_trg must be exclusive, even if not used! If scannerparams.pcindex_stopscan is equal to either of the other pcindex parameters then it will be disabled.

3.2.3. Photocell Triggered Mode (PCTM) II (*scannerparams.mode = 2*)

This mode is very similar to the PCTM I. Unlike PCTM I this mode will deliver user defined sets of segments, regardless of target being detected or not.

Again, start of data collection is triggered by a user defined photocell input, yet terminated when user specified number of segments, *scannerparameters.segments = i* are acquired. After receiving the last segment the scanner will not buffer or send any more data unless given photocell triggers scanning and new set of segments is going to be collected. Example: User will request set of 5 segments of 250 scan frames each. User parameters will be set as follow:

```
scannerparams.lead_spots   = na
scannerparams.lead_wait   = na
scannerparams.trail_spots  = na
scannerparams.trail_wait  = na
scannerparams.history      = na
scannerparams.mode         = 2
scannerparams.pcindex_trg = 0
scannerparams.maxwidth    = 250
scannerparams.escale       = 1
scannerparams.segments     = 5
scannerparams.trail_holdscan = 5
```

*scannerparams_pcindex_len and scannerparams.pcindex_trg must be exclusive, even if not used!

Using these parameters, every time the photocell[0] transitions from light to dark, 5 consecutive segments of 250 scan frames will be delivered to the client CPU. Board_Ready event will be issued to the client application after each segment is completed. After all 5 segments are complete the scanner will not deliver any more data unless the photocell[0] transitions from light to dark again.

3.2.4. Continuous Scanning Mode (CSM) (*scannerparams.mode = 3*)

This mode of employment is quite rudimentary and relies completely on the application to analyze data and put together target/log models. Upon startup the scanner will start delivering scan frames on each encoder pulse regardless of direction. The application, nbllib.dll receives data in sections of *maxwidth* size and issues *Board Ready Event* after each of them. Upon each *Board Ready Event* the client application, optimizer transfers the section data, using *nbGetScannedObject()* API call into its own data space and executes data processing. Meanwhile another data section is being collected.

Example:

```
scannerparams.mode         = 3
scannerparams.maxwidth     = 250
```

*scannerparams_pcindex_len and scannerparams.pcindex_trg must be exclusive, even if not used!

3.2.5. Software Triggered Mode (SMT) (*scannerparams.mode = 4*)

Software triggered mode operates exactly as the PCTM II, except the trigger mechanism is an application TCP/IP command rather than a photocell input signal. Upon issuing nbStartScan() API command, NPH will start collecting and subsequently transmitting data, scan frames regardless of target being or not in the scope of sensors. After each segment of *maxwidth* it will issue Board_Ready event to the user application and continues to collect next segment until number of segments specified by user is collected and delivered to the client application. At the same time, user has an option to abort collection of a segment, or force end of the segment, by issuing nbStopScan() API call. This will result in 'forced' termination of current segment of scan frames and resets the NPH to wait for another nbStartScan() command. Using this mode the user is responsible for decision when to request scans and if and when to terminate scan frames acquisition intentionally.

Example:

```
scannerparams.mode      = 4
scannerparams.maxwidth  = 250
scannerparams.segments  = 5
```

*scannerparams.pcindex_len and scannerparams.pcindex_trg must be exclusive, even if not used!

In above example user requests a set of 5 segments of scan frames data, each 250 scan frames wide. Upon software command nbStartScan() NPH will deliver all 5 segments, issuing Board_Ready event after each of them. Once the set is completed, the scanner will not start scanning unless a new, explicit nbStartScan command is issued. nbStopScan() command can be used in cases when the transport mechanism stopped for some reason, thus no more encoder pulses are issued, yet user has not received some outstanding data. At this point user can issue nbStopScan() command that will force the last segment to be completed and the last Board_Ready event will be raised.

3.3. NPH Inputs/Outputs

3.3.1. Sensor [Input/Output] - NPH provides DC power and proprietary high speed differential serial link of up to 24 installed sensors.

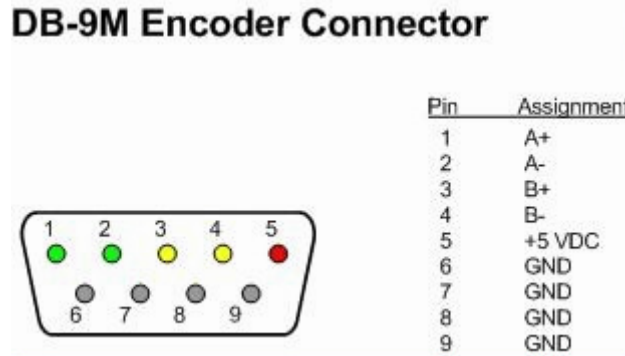
MB Station Input:
The connection to the MB Station is a **Amphenol, C091 61G008 110 2**



Cable:
The power cable connector used is a **DIN 41 326 Standard 8 pin** circular connector



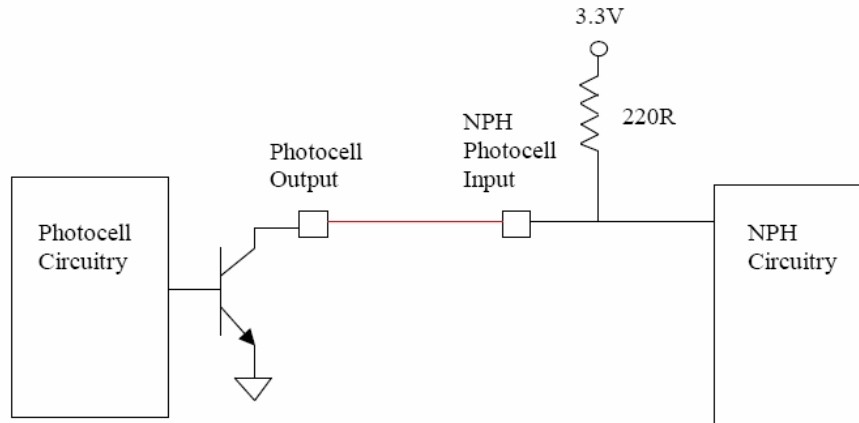
3.3.2. Encoder [Input] - Differential quadrature opto-mechanical encoder maximum recommended resolution 1,500 Hz. DB-9F - cable end DB-9M NPH front face. The NPH will supply the required voltages. No external input is required for powering the encoder as the +5V and GND pins are outputs to be used in the connection of your encoder.



3.3.3. Photocells [Input]

Photocells generally have an open-collector transistor output that operates as an on/off switch. Each photocell input on the NPH has a corresponding internal pull-up resistor that allows the photocell switch to generate 0 or 3.3V in the NPH.

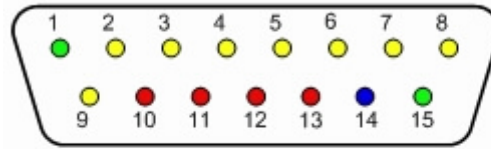
The following diagram shows the connection of the photocell to the NPH circuitry:



When you have a target in range of the photocell then the input pin should be left open, otherwise when no target is in range the photocell should connect the input pin on the NPH to ground.

*Photocell input **Pins 2 – 9** correspond to the photocell **inputs 0-7**. The state of any unused photocell inputs doesn't matter (They can be either grounded or left open).

DB-15M Photocell Input Connector

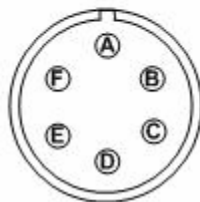


Pin	Assignment
1	Ground
2	Photocell input
3	Photocell input
4	Photocell input
5	Photocell input
6	Photocell input
7	Photocell input
8	Photocell input
9	Photocell input
10	UDP packet send output signal
11	Encoder pulse output signal
12	Reserved
13	Reserved
14	End of Board event output signal
15	Ground

* Using a logic analyzer service technician can monitor output signals on **pin 10** indicating UDP packets being transmitted to the client and **pin 11** for presence of the encoder pulse. The NPH will supply the required voltages.

3.3.4. Power [Input]

Main NPH Power Connector Pinout



A	+15 to +24 V DC (power to the sensors)
B	GND
C	+15 to +24 V DC (power to the sensors)
D	GND
E	Case
F	+15 to +22 V DC (controller power)

3.3.5. Ethernet [Input/Output] - RJ-45 CAT5 Ethernet cable connection to the client computer.

3.3.6. Serial RS-232 [Input/Output] - standard DB-9M connector.

3.4. Scanner System Power Specifications

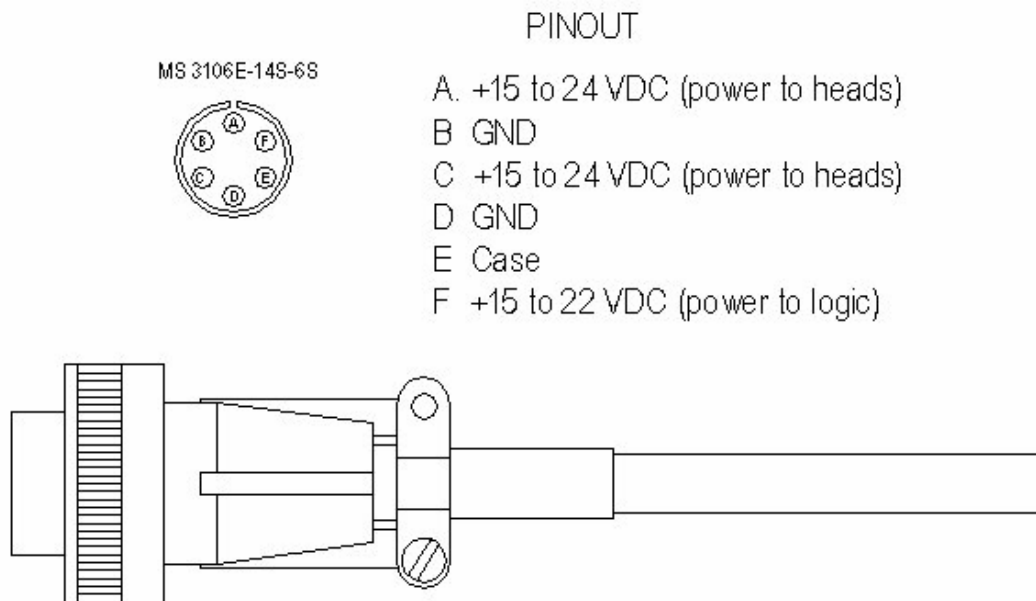
The NetPowerHub concentrator distributes individually fused DC power to each installed sensor as well as provides both proprietary high speed and RS-485 serial communication. The NetPowerHub must be mounted inside an industrial NEMA enclosure at the scanner frame to protect the electronics and is mounted on the back panel of that enclosure.

Power supplies to operate the sensors should be mounted in a cabinet on or near the scanner frame to reduce voltage drops to each sensor. A single four pair plenum grade (i.e. industrial) shielded CAT5 cable is run to each sensor from the NPH-66. The power cable connector used is a DIN 41 326 Standard 8 pin circular connector.

Name	Description
VDC+/-	DC power (VDC+) and Ground return (VDC-)
RxTx+/-	Asynchronous 19.k KBaud serial bi-directional pair(RS-485)
SerClk+/-	High speed Serial Clock pair - Future upgrade path to eliminate fiber optics and M24-style concentrator
SerDat+/-	High Speed Serial Data pair - Future upgrade path to eliminate fiber optics and M24-style concentrator

Each of the four twisted pairs in the CAT5 cable is connected to one of the RS-485 differential or power/ground (+/-) pairs. The B8A sensor has been specially designed to work with high source resistance power supplies, thus allowing more inexpensive power cabling to be used (such as CAT5 cable). Note that the use of CAT5 cable assumes a 24V supply, and a maximum cable length of 50 feet. The CAT5 cable DC resistance is about 25 ohms per thousand feet, or about 2.5 ohms for the 100 foot round trip on the 50 foot cable. The B8A sensor draws about 800 mA at 24 volts, so a 1.5V drop occurs on the cable. Lower voltages (such as 15V) cause the switching power supplies in the B8A sensor to draw more current, and will make the voltage drop on the cable larger. The NetPowerHub alone draws 500mA @ 24V min (12Watt).

Cable Pinout for NPH DC Power Input



4. NPH API Functions

The functions described in this section provide programming interface between the client computer and the NPH using NBLIB.DLL LMI API library. Communication uses TCP/IP and UDP protocols over the Ethernet network connection between the NPH unit and the CPU. A typical client computer setup will have an Ethernet network adapter installed with Windows NT 4.0 / 2000 / XP operating system and networking services.

nbllib.dll is a C-language compatible API. All of the functions described in this section are declared in the *nbllib.h* header file. See one of the following topics for more information:

4.1. NbLib API Functions

<u>nbOpenScanner</u>	Initializes the ethernet interface.
<u>nbCloseScanner</u>	Terminates the ethernet interface.
<u>nbGetLaserStatus</u>	Returns individual laser status (disabled or enabled).
<u>nbSetLaserStatus</u>	Disable/enable laser readings.
<u>nbGetLaserInfo</u>	Returns diagnostic information from a sensor.
<u>nbGetHeadInfo</u>	Returns individual sensor diagnostics.
<u>nbGetEncoderInfo</u>	Returns the current encoder count.
<u>nbSetEncoderInfo</u>	Sets encoder counter.
<u>nbGetPhotocells</u>	Retrieves real-time photocells status.
<u>nbGetPhotocellHistory</u>	Retrieves photocell history buffer.
<u>nbGetScannedObject</u>	Detects and collects scanned object / board.
<u>nbGetRanges</u>	Returns real-time range readings.
<u>nbGetAllRanges</u>	Returns real-time range readings from all the installed sensors.
<u>nbGetVersion</u>	Returns API library software version.
<u>nbGetFwVersion</u>	Returns NPH-66 code version.
<u>nbGetFPGAVersion</u>	Returns FPGA code version.
<u>nbGetLastAsyncEvent</u>	Returns the last asynchronous event information.
<u>nbGetOffsets</u>	Read current system calibration data.
<u>nbSetOffsets</u>	Set system calibration data.
<u>nbGetSensorMap</u>	Returns a mask of attached sensors.
<u>nbSetSensorMap</u>	Set the map of attached sensors.
<u>nbGetLastError</u>	Retrieve last API error.
<u>nbGetValidLasers</u>	Read number of lasers given sensor has.

<u>nbSetValidLasers</u>	Specify number of lasers given sensor has.
<u>nbSensorReset</u>	Reset individual sensor.
<u>nbUploadUserParameters</u>	Upload user parameters to the NPH-66 concentrator.
<u>nbGetSystemData</u>	
<u>nbReadLogLength</u>	Upon ASYNC_EVENT_LOG_PC_INFO event read given photocell transitions record.
<u>nbResetNPH</u>	Client command to reset/reboot NPH-66 unit.
<u>nbSerialSettings</u>	Initialize NPH-66 COM port.
<u>nbStartScan</u>	Software trigger of scan frames acquisition upon each encoder pulse.
<u>nbStopScan</u>	Abort/complete data acquisition initiated by nbStartScan.
<u>nbSetAllLasers</u>	

BOOL nbOpenScanner(OPENSCANNERST *opp, SCANNERPARAMS *user_sp);

This function initializes Ethernet communication between the client CPU and the NPH-66. This function must be called prior making any other calls to **nblib.dll** API library.

Input

<u>OPENSCANNERST</u> <i>*ops</i>	Pointer to a structure of type OPENSCANNERST. This structure contains two parameters required to initialize the NBLIB for board scanning: <i>opst.IPAddress</i> ASCII string, specifying the host IP address, such as "192.168.0.10". If this field is initialized as NULL, the NBLIB will detect the host. Specifying the host address is only necessary for the unlikely cases when there are more than a single host available at the same time. <i>opst.hEvent</i> Contains a handle to an event. If this handle is valid, the library will signal all asynchronous events using this handle. It is the caller's responsibility to create manual resetable event by a call such as: <code>opst.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);</code> If this field is set to NULL, the library will not issue any <u>asynchronous</u> event notifications.
<u>SCANNERPARAMS</u> <i>*user_sp</i>	pointer to the structure containing user parameters for the scanner operation.

Return Value

BOOL bReturn	TRUE on success. FALSE on error. Call <u>nbGetLastError</u> to retrieve more information on the nature of the
--------------	---

failure.

See: nbCloseScanner

BOOL nbCloseScanner();

This function terminates communication with the host and performs any necessary cleanup.

Input

VOID

Return

BOOL TRUE on success
 FALSE on error. Call nbGetLastError API function to retrieve more information on the nature of failure.

Remarks

If nbOpenScanner returned TRUE when called to initialize the host communication, nbCloseScanner should be called to perform any necessary cleanup. If this function is not called, resource leaks may result.

BOOL nbSensorReset(int sensor_address);

Soft reset a sensor. This call will broadcast a reset command to all installed sensors. There is no reply to this command.

Input

int sensor_address The sensor's logical address.

Return

BOOL bResult TRUE on success
 FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

Remarks

Current release ignores the input and broadcasts reset command to all installed sensors. All the installed sensors will reset. This takes approximately 5 sec.

BOOL nbGetLaserStatus(int sensor, int laser, BOOL *enable);

This function is called to determine whether the library is currently using the indicated laser in acquiring board scan data.

Input

int sensor The sensor index of which laser status should be returned.

int laser	The laser index of the laser whose status should be returned.
BOOL *enable	Returns current status of the specified laser. A value of TRUE indicates that the spot is currently used in acquiring board scan data. A value of FALSE indicates that the laser readings are disabled and reported range value is always Out of Range.

Return

BOOL bResult	TRUE on success FALSE on error. Call nbGetLastError API function to retrieve more information on the nature of failure.
--------------	--

Remarks

All laser spots for any sensor that is not available will be implicitly disabled. If any lasers are known to return spurious readings, they can be disabled with a call to [nbSetLaserStatus](#). Disabling a laser spot reading has no effect on the actual sensor but indicates to the library that the range data from the laser should not be used/ignored.

See Also [nbSetLaserStatus](#)

BOOL nbSetLaserStatus(int sensor, int laser, BOOL enable);

This function is called to specify whether the library should be using the indicated laser in acquiring board scan data.

Input

int sensor	The sensor index housing given laser.
int laser	The laser index of the laser whose status we want to modify.
BOOL enable	Desired status for the specified laser. A value of TRUE indicates that the laser should be used in acquiring board scan data. A value of FALSE indicates that the laser spot reading will be always Out of Range value.

Return

BOOL bResult	TRUE on success FALSE on error. Call nbGetLastError API function to retrieve more information on the nature of failure.
--------------	--

Remarks

When the library is initialized, all lasers for all sensors are set to be enabled. If any lasers are known to return spurious readings, they can be disabled with a call to this function. Disabling a spot has no effect on the actual sensor but indicates to the library that the data from the laser spot readings should not be used when acquiring board scans.

See also [nbGetLaserStatus](#)

BOOL nbGetValidLasers(int sensor, int *numlasers);

This function is called to retrieve number of valid lasers setting for a given sensor. These settings are permanently stored in the NPH concentrator and can be modified using [nbSetValidLasers\(\)](#) function.

Input

int sensor	The logical/system sensor index of the desired sensor.
int *numlasers	Pointer to a destination. Depending on a sensor attached return value can be 0(all the lasers are disabled), 8 for B8 & B8A sensors, 16 for B16 sensor or 23 for M24 sensor.

Return

BOOL bResult	TRUE on success; FALSE on error. Call nbGetLastError to retrieve more information on the nature of the failure.
--------------	--

Remarks

M24 sensors are not supported at this time.
See also [nbSetValidLasers\(\)](#) library function.

BOOL nbSetValidLasers(int sensor, int numlasers);

This function is called to specify number of lasers a given sensor has. This value is then permanently stored in the NHP concentrator.

Input

int sensor	The logical/system sensor index of the desired sensor.
int numlasers	Number of lasers for given sensor type. Specify 8 for B8A sensor, 16 for B16 sensor, or 23 for M24B sensor type.

Return

BOOL bResult	TRUE on success; FALSE on error. Call nbGetLastError to retrieve more information on the nature of the failure.
--------------	---

Remarks

M24 sensors are not supported at this time.
See also [nbGetValidLasers\(\)](#) library function.

BOOL nbGetLaserInfo(int sensor, int laser, BXSPOTDATA *laserData);

This function is called to retrieve diagnostic data for a particular laser.

Input

int sensor The logical/system sensor index of the desired sensor.

int laser Laser index of a given sensor.

[BXSPOTDATA](#)
*laserData

Pointer to a structure of type BXSPOTDATA that is to be filled in with diagnostic data read from the sensor.

Return

BOOL bResult TRUE on success;
FALSE on error. Call [nbGetLastError](#) to retrieve more information on the nature of the failure.

BOOL nbGetHeadInfo(int sensor, BXHEADDATA *sensorData);

This function is called to retrieve diagnostic data for a particular sensor.

Input

int sensor The logical/system sensor index of the desired sensor.

[BXHEADDATA](#)
*laserData

Pointer to a structure of type BXHEADDATA that is to be filled in with diagnostic data read from the sensor.

Return

BOOL bReturn TRUE on success;
FALSE on error. Call [nbGetLastError](#) to retrieve more information on the nature of the failure.

BOOL nbGetEncoderInfo(unsigned int *encoderCount);

This function writes current encoder count into the pointer location.

Input

unsigned int Pointer to the destination variable.
*encoderCount

Return

BOOL bResult TRUE on success;
FALSE on error. Call [nbGetLastError](#) to retrieve more information on the nature of the failure.

Remarks

When the NetPowerHub is first powered-on, the encoder count is initialized to zero. The encoder count is then incremented with each received encoder pulse until a counter overflow occurs. The encoder count is then zeroed and begins counting up again.

See Also [nbSetEncoderInfo](#)

BOOL nbSetEncoderInfo(unsigned int encoderCount);

This function resets the encoder counter to desired value.

Input

unsigned int User value to replace the current encoder count.
encoderCount

Return

BOOL TRUE on success;
bReturn FALSE on error. Call [nbGetLastError](#) to retrieve more information on the nature of the failure.

Remarks

When the host is first powered-on, the encoder count is initialized to zero. The encoder count is then incremented with each received encoder pulse until a counter overflow occurs. The encoder count is then zeroed and begins counting up again. Using this function prior any API calls that use encoder can prevent the encoder from wrapping around.

See also [nbGetEncoderInfo](#)

*BOOL nbGetPhotocells(PHOTOCELLST *photocellsinfo);*

This function fills a user buffer with the current photocell information and current encoder count. When the NPH is first powered-on, the encoder count is initialized to zero. The encoder count is then incremented with each received encoder pulse until a counter overflow occurs. The encoder count is then zeroed and begins counting up again.

Input

[PHOTOCELLST](#) user buffer to be filled with the current photocells value and encoder count.
*photocellsinfo

Return

BOOL bResult TRUE on success
 FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

See Also [nbGetPhotocellHistory](#)

*BOOL nbGetPhotocellsHistory(PHOTOCELLST *photocellsbuffer, int buffersize);*

This function fills a user buffer with the current photocell information and current encoder count. At each encoder tick the value of photocells is recorded by the host in a ring buffer. The most recently recorded data may be recovered using this API. The NPH buffer can hold at least MAX_PHOTOCELL_BUFFERS readings. An attempt to read more data than the buffer contains will lead to an error.

Input

[PHOTOCELLST](#)
*photocellsbuffer user buffer to be filled with the most recent photocells values and encoder counts.

int buffersize number of elements in photocellsbuffer.
[1..[MAX_PHOTOCELL_BUFFERS](#)]

Return

BOOL bResult TRUE on success
FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

See Also [nbGetPhotocells](#)

*BOOL nbGetRanges(int sensor, int numsamples, unsigned short *ranges);*

This function fills a user buffer with real-time range readings for all the lasers within given sensor as read by the NetPowerHub.

Input

int sensor The sensor index.

int numsamples Number of range samples to read.

unsigned short
*ranges User buffer to be filled with the ranges.

Return

BOOL bResult TRUE on success
FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

Remarks

The buffer must be sufficient enough to hold all requested range readings. A recommended calling sequence should be:

```
buffer = (short*)malloc(sizeof(short)*NB_MAXSPOTS);
```

```
if (nbGetRanges(head,NB_MAXSPOTS,buffer))
```

```
    {  
        // Process ranges  
    }  
else  
    {  
        // Process error  
    }
```

BOOL nbGetAllRanges(int numHeads, int numSpots, int *isAvailable, unsigned short *ranges);

This function is called to acquire one time range readings from all the sensors in the system. Note that the ranges for any sensor that is not available will be set to LMI_OUTRANGE.

Input

int numHeads	The number of sensors in the system (0..NB_MAXHEADS)
int numSpots	Number of spots for each sensor (0..NB_MAXSPOTS)
unsigned int *isAvailable	Reports whether the sensor is present
unsigned short *ranges	User buffer to be filled with the ranges.

Return

BOOL bResult	TRUE on success FALSE on error. Call nbGetLastError API function to retrieve more information on the nature of failure.
--------------	--

Remarks

The isAvailable buffer must contain at least numHeads elements. The ranges buffer must contain at least numHeads*numSpots elements

A recommended calling sequence should be:

```
isAvailable = (int*)malloc(sizeof(int)*numHeads);  
ranges      = (unsigned short*)malloc(sizeof(unsigned short)*numHeads*numSpots);  
  
if (nbGetAllRanges(numHeads, numSpots, isAvailable, ranges))  
{  
    for (i = 0; i < numHeads; ++i)
```

```
        {
            if (isAvalible[i])    //was this sensor present in the
                                // system?
            {
                for (j = 0; j < numSpots; ++j)
                {
                    currentRange = ranges[i][j];
                    //do something with the range data
                }
            }
        }
    }
else
    {
        //process error
    }
}
```

BOOL nbGetScannedObject(SCANNEDOBJECTST *object);

This high level function gathers range readings for all the lasers into a user provided buffer. The function detects the leading and trailing edge of an object (board) being scanned and realigns all individual laser range data using the system offset table obtained by user during system calibration. Each entry in a laser range data buffer corresponds to an encoder pulse.

Input

[SCANNEDOBJECTST](#) Pointer to a user allocated structure containing the scan buffer of the
*buffer whole object / board.

Return

BOOL bResult TRUE on success
FALSE on error. Call [nbGetLastError](#) function to retrieve more information on the nature of failure.

Remarks

This API is a blocking call with a timeout. This function will attempt to gather scan and photocells information for each encoder interrupt and store it in the user buffer, line by line. The user must allocate the buffer to hold enough scan lines using a call such as:

```
buffer = (SCANNEDOBJECTST*)malloc(sizeof(SCANNEDOBJECTST)+
maxscanlines*sizeof(ONELINEST));
```

If the buffer is successfully allocated, the field allocated lines must be initialized:

```
if (buffer != NULL)
```

```
    buffer->allocatedlines = maxscanlines;
```

```
else
```

```
    // ....process error
```

After the buffer has been successfully allocated and initialized, we can call the API:

```
if (nbGetScannedObject(buffer))
{
    // ...process scanlines and photocells
}
else
{
    // ...process error

    error = nbGetLastError(NULL);
}
```

Upon successful completion of this function, the SCANOBJECTST fields are filled as follows:

<i>buffer.firstscanline</i>	holds index of the buffer line with the first board scan information.
<i>buffer.lastscanline</i>	holds the index of the buffer line with the last board scan information.
<i>buffer.firstphotoline</i>	holds index of the buffer line with the first valid photocells information.
<i>buffer.lastphotoline</i>	holds index of the buffer line with the last valid photocells information.

All additional fields are for internal use only and should be ignored by the user. Each valid scan line contains encoder position, photocells value and the actual scan data. The scan data is adjusted using the internal offsets table. (See nbSetOffsets for more information).

This function can generate errors specific to this API:

NB_ERROR_API_TIMEOUT	function timed out waiting for scans
NB_ERROR_NO_BEGINNING	function failed to detect the leading edge
NB_ERROR_NO_END	function failed to detect the trailing edge before the user buffer was filled.
NB_ERROR_NO_BEGINNING_NO_END	function failed to detect both the leading and the trailing edge.

NB_ERROR_PACKETS_MISSING	Some UDP packets send by the host were lost.
NB_ERROR_DATA_MISSING	Scan data missing, typically caused by the encoder being too fast.

```
int nbGetVersion(void);
```

Report the version number of the **nblib.dll** software library.

Input

void

Return

int version	The current library version
-------------	-----------------------------

```
int nbGetLastAsyncEvent(ASYNCEVENTST *apt);
```

This function returns the last asynchronous event information.

Input

<u>ASYNCEVENTST</u> *apt	Pointer to an ASYNCEVENTST structure that will be modified with current information.
---	--

Return

int code	The code for the last asynchronous event.
----------	---

Remarks

The user is optionally notified of some asynchronous events (see `nbOpenScanner`). If an event is signaled, user can retrieve relevant information using this function. User will typically start a separate thread that waits for an event to be signaled:

```
hEventThread
StartReadingThread((LPTHREAD_START_ROUTINE)EventWaitThread);
The thread itself is an infinite loop waiting for and processing all events:
void EventWaitThread()
{
    ASYNCEVENTST aest;

    while (1)
    {
        WaitForSingleObject(ghEvent, INFINITE);
        nbGetLastAsyncEvent(&aest);

        switch(aest.code)
        {
            case WAIT_OBJECT_0:
                // Event signaled, process it
                break;
            case WAIT_TIMEOUT:
                // Event timed out, no action
                break;
            case WAIT_FAILED:
                // Error, handle it
                break;
        }
    }
}
```

```
        {
            case ASYNC_EVENT_CONNECTION_LOST:
                // process connection lost...
                break;
            case ASYNC_EVENT_HEADS_CHANGED:
                // process heads change. Some sensor died (or came
back to life!)
                HeadsMask = aest.data; //bitfield of active heads
                break;
            case ASYNC_EVENT_BOARD_READY:
                result = nbGescannedObject(&object);
                break;
        }
        // allow event again
        ResetEvent(ghEvent);
    }
}
```

The code above assumes the variable ghEvent contains the same event handle that was passed to nbOpenScanner().

Asynchronous event types:

```
#define ASYNC_EVENT_CONNECTION_LOST    0x00000001
#define ASYNC_EVENT_HEADS_CHANGED      0x00000002
#define ASYNC_EVENT_BOARD_READY        0x00000004
```

```
typedef struct
{
    int code;
    int data;
} ASYNCEVENTST;
```

***BOOL LMIAPI nbGetSystemData(BXHEADANDSPOTDATA
*systemData, int *numHeads, int numSpots);***

This function is called to acquire both the sensor and the spot diagnostic information from all sensors in the system.

Input

[BXHEADANDSPOTDATA](#)

*systemData

pointer to the destination

int *numHeads

the number of elements in the systemData array
(0..NB_MAXHEADS)

int *numSpots

the number of laser spots for each sensor in the system
(0..NB_MAXSPOTS)e.g. 8 for a B8A

Return

TRUE on success
FALSE on error.
Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

Remarks

This function may require a long time (on the order of one second) to complete. Do not use this function in time-critical code that requires rapid, deterministic completion.

*BOOL nbSetOffsets(int *offsets, int tablesize);*

This function is called to specify the laser offsets table used implicitly by [nbGetScannedObject](#). The table must have one entry for each spot for all sensors. User can provide his/her own offsets table that has been calculated to compensate for sensors mounting misalignments.

Input

int *offsets Pointer to the offsets buffer(table).

int tablesize Size of the offsets table in bytes. Must be
 sizeof(int) *NB_MAXHEADS*NB_MAXSPOTS
The table must be of full size even if not all sensors are installed.

Return

BOOL bResult TRUE on success
 FALSE on error.
Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

See Also [nbGetOffsets](#)

*BOOL nbGetOffsets(int *offsets, int tablesize);*

This function is called to obtain the current system calibration laser offsets table. The table is implicitly used by [nbGetScannedObject](#) API call.

Input

int *offsets Pointer to the destination table of laser offsets.

int tablesize Size of the offsets table in bytes. Must be at least:
 sizeof(int) *NB_MAXHEADS*NB_MAXSPOTS
The table must be of full size even if not all sensors are installed.

Remarks The default sensor map maps connector indices into logical order as one to one:

```
int SensorMap[NB_MAXHEADS] =
{
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
};
```

The SensorMap is used internally by all APIs dealing with sensor data.

See also [nbGetSensorMap](#)

int nbGetLastError(const char **description);

This function may be called to retrieve an error code after a function in the *nbllib.dll* library returns unsuccessfully.

Input

<i>const char **description</i>	If not NULL, the library will return a pointer to an ASCII string describing the last error.
-------------------------------------	--

Return

<i>int lasterror</i>	Numeric code of the last error.
----------------------	---------------------------------

Remarks

If a function in the NBLIB library returns unsuccessfully, call this function immediately to determine the cause of the error. If any calls are made to the NBLIB library between the unsuccessful call and the call to this function, the error information will most likely be lost.

The return value of nbGetLastError is undefined when the most recent call to the NBLIB library returned successfully

BOOL nbUploadUserParameters(SCANNERPARAMS *user_sp);

This function is called to initialize or modify application specific parameters for the Board Detection Logic (BDL) residing in the NPH memory.

Input

SCANNERPARAMS <i>*user_sp</i>	Pointer to the NPH concentrator parameters structure.
--	---

Return

BOOL bResult	TRUE on success FALSE on error. Call nbGetLastError API function to retrieve more information on the nature of failure.
--------------	--

Remarks

Upon boot up the NPH concentrator initializes BDL to following default parameters:

* MINSPOTS and MINWAIT are applied to both leading and trailing edge of a target/board

```
#define MINSPOTS    3           // minimum number of laser spots seen to
validate a target
#define MINWAIT     5           // minimum encoder pulses to hold a state
#define MAXREVERSE  192        // allow maximum 6" reverse movement (@
1/32 encoder resolution)
#define MAXBOARD    1024       // default board size in encoder pulses
#define PADDING     10        // bit of a history prior to start of
board - is sent to client as well
```

See also [SCANNERPARAMS nbGetScannedObject\(\)](#)

BOOL nbGetFwVersion(char *nbfw_version);

Report the version number of the **NPH** firmware code.

Input

char *nbfw_version	pointer to a destination character string. The allocated string size must be minimum MINVERSIONSZ characters.
--------------------	--

Return

BOOL	TRUE on success FALSE on error. Call nbGetLastError API function to retrieve more information on the nature of failure.
------	--

BOOL nbReadLogLength(PCSTATEST *pcstates, int source);

This function returns encoder values recorded at given photocell transitions. NPH logs current encoder count on each photocell transition, light to dark and vice versa. Upon initialization of the scanner user specifies the photocell index to be used for this data logging. Once the transition occurs NPH will transmit PCSTATEST structure to the client computer over the Ethernet connection using UDP protocol. Upon receiving this UDP packet the nlib.dll will issue the ASYNC_EVENT_LOG_PC_INFO event. Subsequently user then issues this call to access the information.

This information can be retrieved anytime using a TCP/IP command by passing value **1** in place of *source* argument.

Input

[PCSTATEST](#) *pcstates Structure containing photocell transition encoder counts

int source 0 - uses the most recent UDP information received from the NPH
 1 - issue the *TCP/IP request to poll for the data from the NPH

Return

BOOL bResult TRUE on success
 FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

* This method is somewhat redundant since the information is being received automatically over the UDP protocol. Yet, there may be an occasion when user may need to use this call, such as chain not moving at that time or a confirmation is needed.

BOOL nbStartScan(void);

Sends a TCP/IP command to the NPH unit to trigger collection of scan frames upon each encoder pulse up to *maxwidth* scan frames. This command is recognized by NPH unit only in the Software Trigger Mode (see [Modes of Operation](#)). User must specify the direction of the transport (useful for rotation scanning in both directions)

Input

int direction encoder direction LMI_ENC_FWD or LMI_ENC_REV

Return

BOOL bResult TRUE on success
 FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

Remarks

See also [nbStopScan\(\)](#)

BOOL nbStopScan();

This call is used in Software Trigger Mode only (see [Modes of Operation](#)). Upon issuing this TCP command the NPH will terminate data acquisition of current segment, subsequently issues the Board_Ready event regardless of number of scan frames acquired so far. It is intended to be used in case the transport stops, thus no more encoder pulses are issued and the last portion of scan frames is not available to the user.

Input

void

Return

BOOL bResult TRUE on success
 FALSE on error. Call [nbGetLastError](#) API function to retrieve more
 information on the nature of failure.

Remarks

See also [nbStartScan\(\)](#)

```
BOOL nbSetAllLasers(int cmd);
```

Sends a TCP/IP command to turn OFF/ON all the lasers in the system. This call is useful for the maintenance of the scanner.

Input

```
int cmd      LMI_LASERS_ON
             LMI_LASERS_OFF
```

Return

BOOL bResult	TRUE	on	success
	FALSE	on	error.
	Call nbGetLastError API function to retrieve more information on the nature of failure.		

Remarks

This command requires proper COM port settings for the particular sensor model.

```
BOOL LMIAPI nbSerialSettings(int baudrate,int databits, int stopbits, int
parity);
```

This function is called to initialize a serial port on the NPH-66 to enable serial communication between the NPH-66 and individual sensors.

Input

int baudrate	Desired COM baudrate
int databits	8
int stopbits	1
int parity	None

Return

BOOL bResult TRUE on success
 FALSE on error. Call [nbGetLastError](#) API function to retrieve more information on the nature of failure.

Remarks

This command requires proper COM port settings for the particular sensor model.

void nbResetNPH(void);

Soft reset the NPH-66 unit.

Input

void

Return

void

Remarks

This command is used to Reset the NPH

4.2. NbLib API Errors

NB_NOERROR • No error has occurred	0
NB_ERROR_SOCKET	1
NB_ERROR_NOHOST No host is detected. Ensure that your NPH is turned on	2
NB_ERROR_WRONG_SOCKET_VERSION	3
NB_ERROR_INVALID_FPGA_FILE • Re-upload firmware	4
NB_ERROR_READING_FPGA_FILE • FPGA corrupt, Re-upload firmware	5
NB_ERROR_NO_MEMORY • No storage has been allocated	6
NB_ERROR_UNSUPPORTED_FPGA_FILE Check firmware version, Re-upload firmware	7
NB_ERROR_OPEN_FPGA_FILE • Re-upload firmware	8
NB_ERROR_NOT_CONNECTED • Connection between PC and NPH lost	9
NB_ERROR_MISSING_DIAGS • Missing diagnostic data. No sensor head connected or connection has been lost	10
NB_ERROR_INVALID_PACKET	11
NB_ERROR_INVALID_CMD	12
NB_ERROR_CMD_TIMEOUT	13
NB_ERROR_INVALID_HEAD • Invalid sensor head selected (Head selected > 24, or no head connected to port)	14
NB_ERROR_INVALID_SPOT • Invalid spot selected (spot selected > MaxSpots, or no sensor head connected to port)	15
NB_ERROR_INVALID_CMD_REPLY	16
NB_ERROR_HEAD_NOT_PRESENT • No head connected to port	17
NB_ERROR_NO_BUFFER • No storage has been allocated	18
NB_ERROR_API_TIMEOUT • Connection to Firmware has been lost	19
NB_ERROR_NO_BEGINNING • failed to detect the leading edge of board	20
NB_ERROR_NO_END • failed to detect the trailing edge of board before user buffer filled	21

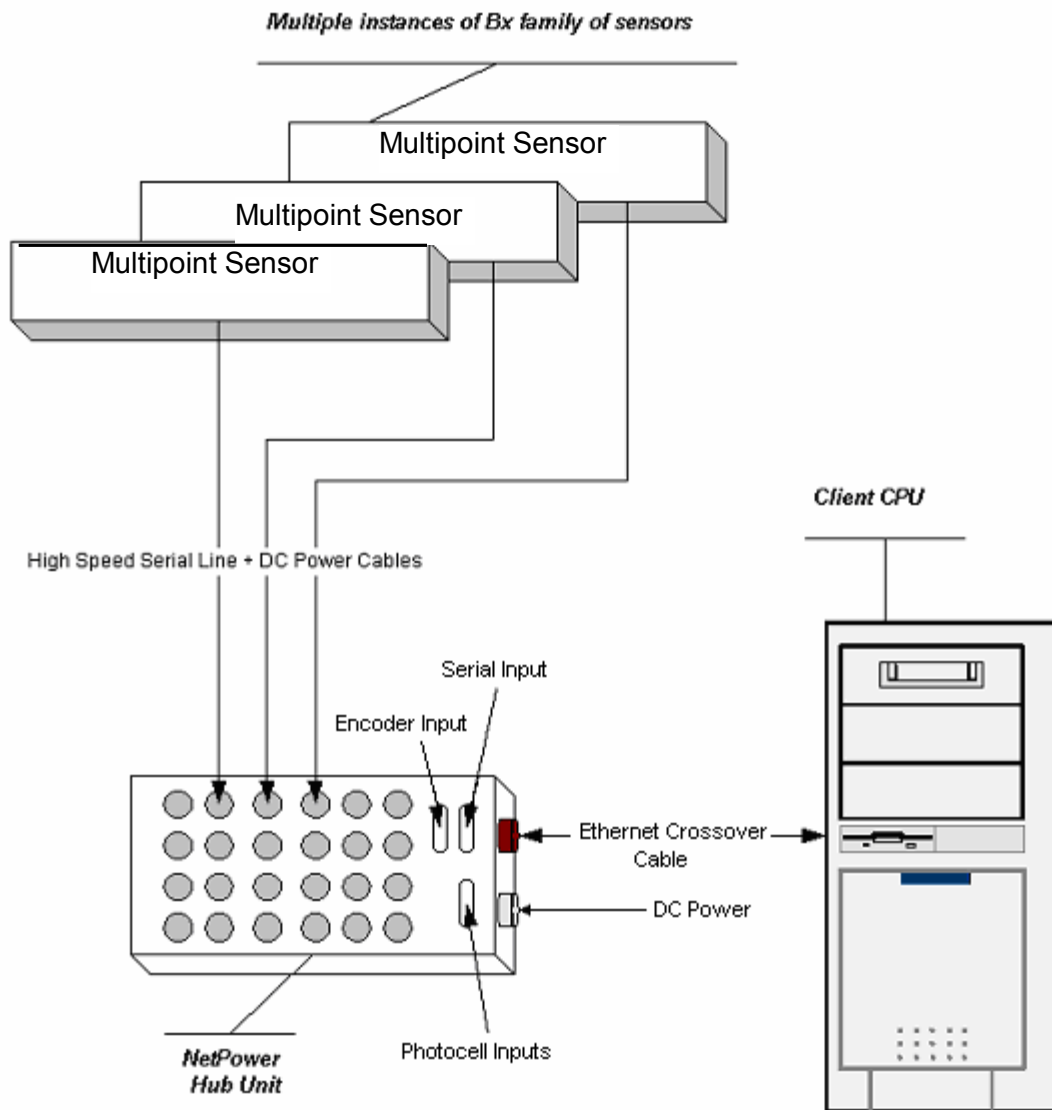
NB_ERROR_NO_BEGINNING_NO_END failed to detect both the leading and the trailing edge	22
NB_ERROR_PACKETS_MISSING Some UDP packets send by the host were lost	23
NB_ERROR_DATA_MISSING - Scan data missing, typically caused by the encoder being too fast, or - too much network traffic	24
NB_ERROR_INCORRECT_ARGUMENT	25
NB_ERROR_INVALID_TABLE	26
NB_ERROR_INVALID_SERIAL_REPLY	27
NB_ERROR_CONNECTION_LOST	28
NB_ERROR_CONNECTION_CLOSED	29
NB_ALREADY_CONNECTED Already connected to a NPH	30
NB_TCP_COMMAND_FAILED	31
NB_ERROR_UDP_THREAD	32
NB_ERROR_BUFFERSIZE Invalid buffer size set. The user maxwidth parameter is set higher than the number of allocated scanlines	33

5. Scanner System

5.1. Scanner System Connection

Multipoint sensor scanning system consists of multiple instances of sensors (Bx00, B8A, B16, or M24B) connected to and powered by a NPH-66 concentrator unit. The NPH unit is then connected to the client computer over the Ethernet network.

Connection Diagram



LMI Scanning System Architecture.

Embedded NPH software provides all the functionality to acquire real-time range measurements from all the installed sensors and communicates these data over the Ethernet communication link to the AIP library (*nbllib.dll*) residing on a client computer. LMI provides the OEM API software support for Windows NT/2000/XP OS platforms.

Note: For best performance connect directly from the NPH to your PC via crossover Ethernet cable.

5.2. Warnings and system design considerations

When designing your B-Series scanner system there are a few points that must be taken into consideration to avoid any scanning problems.

- At no time should welding take place near the scan frame. If it is necessary to weld near the frame the scanner system should have the power turned off and disconnected and the sensors should be covered.
- When performing a system calibration you should use an aluminum calibration bar that has been painted with a flat tan colored paint. Also the calibration bar should be handled with care so that no scratches or exposed aluminum is shown
- When mounting the sensors they should be staggered side to side. Also the top sensors should have their lasers facing directly into the laser windows of the bottoms sensors, and vice versa.
- When designing your frame it is important to have all chains running between sensor heads so that the field of view of the sensors is not being disturbed. If this isn't possible then any lasers hitting chains must be mapped out of the system as they will cause spurious data.
- Encoder resolution is a value that should be designed into your system and then verified once the frame is complete. The best way to find encoder resolution is to run your chain until 1500+ encoder pulses are generated, then measure the physical chain travel to at least the nearest 1/16". The encoder resolution will then be the number of pulses over the distance traveled.
- If you are designing a system that requires exact width measurement then we recommend using photocells for the width measurement. This is because the sensors are meant for range measurement and wane detection but not width measurement. Therefore, the sensors will have an error if they are being used for width measurement, but photocells will give a very good accuracy.
- When a sensor is shipped, the logical address is set to the last two digits of the serial number of that sensor as a factory default setting. It is the responsibility of the customer to check and modify the bus address if more than one sensor use the same logical address on one system. Note sensors with last two digits of the serial number that are zeros (e.g. B8001600) are set to 100. See section 5.3, step 16 for bus address change.

5.3. Sensor Firmware Upload Procedure

Tools

Hardware

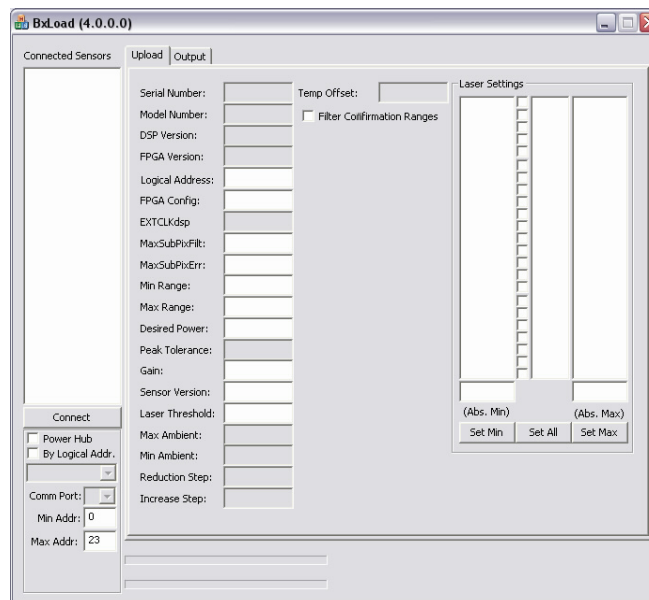
Desktop or a Notebook PC with Ethernet 10/100 Mb adapter installed.
NetPowerHub Concentrator

Software

Windows '9x/NT/2000/XP Operating system
BxLoad.exe
Nbllib.dll
Nbtest.exe
Nbllib.dll

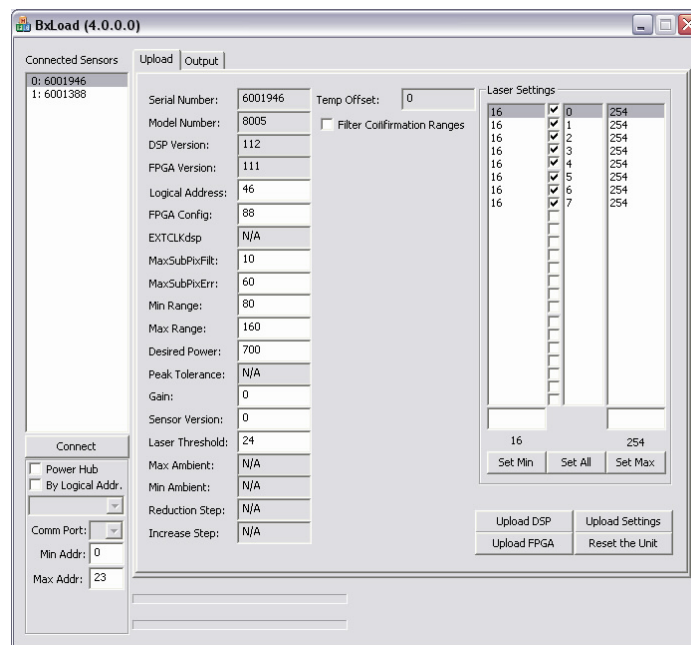
Procedure

1. Connect all the sensors to the NPH-66 unit.
2. Connect NPH-66 to your network using a standard CAT5 Ethernet cable or directly to a computer using a crossover CAT5 Ethernet cable.
3. Power up the NPH-66. The unit will boot up in about 10 seconds and is ready to operate.
4. Run BxLoad.exe utility on PC.
5. Click on the 'Connect' button from the dialogue box (see Fig. 1.0)



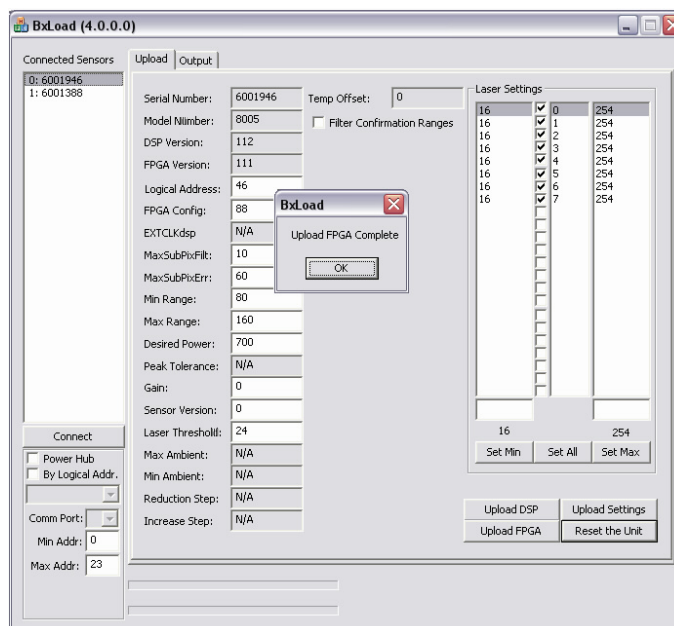
BxLoad - Fig 1.0

6. Program will detect all operational sensors connected to the NPH unit. Once finished list of detected sensors serial numbers will appear in the left pane of the dialogue window (Fig. 1.1)



BxLoad, Connected – Fig1.1

7. From the list select a sensor / serial number you wish to upload code to.
8. To upload new FPGA code to the selected sensor press the Upload FPGA button at the bottom of the dialogue box. Program will display a Windows file selection dialogue box. Select the file to be uploaded (.mcs extension). Upon acceptance the BxLoad program will automatically start uploading selected file into the sensor. Upload progress is indicated by a progress bar. Once finished a message box (Fig.1.2) is displayed. IF THE UPLOAD FAILS DO NOT RESET OR POWER DOWN THE SENSOR, please try the upload again.



BxLoad, Connected – Fig 1.2

Press OK. Program will return to the initial screen (Fig. 1.1)

9. Next press the Upload DSP button (next to the FPGA Firmware button) to upload new DSP code. Similarly to FPGA Firmware upload procedure a file selection dialogue will appear allowing you to select a DSP.LOD file to upload. Select the file and press OK.
10. Next BxLoad will display DSP Parameters dialogue box (Fig. 1.1). These parameters are modified either during initial setup or by a qualified LMI field service technician.
11. To accept the settings and upload them, click on the 'Upload Settings' button. BxLoad will then automatically start uploading code to the sensor. Progress is indicated by a progress bar. Once finished a dialogue message box appears (Fig. 1.2). Press OK. IF THE UPLOAD FAILS DO NOT RESET OR POWER DOWN THE SENSOR, please try the upload again.
12. At this point the selected sensor is uploaded with both new FPGA and DSP code.
13. Click on the 'Reset the Unit' button to reset the sensor so the settings can take effect. You will now return to the initial screen (Fig 1.1) and you can select another sensor to upload.
14. Repeat the procedure, steps 7 through 12 for all the remaining sensors.
15. Once finished you can reconnect and BxLoad should detect all of the installed sensors again and list their serial numbers in the left pane of the dialogue box.
16. Check the logical address of each sensor. Change the logical address of the sensor which has the same logical address of a sensor in the same system. To change the logical bus address of a sensor, you can use digits in between 1 thru 255. In the Logical Address type in the two or three digits preferred, again maximum being 255 and repeat steps 11 and 13.
Note sensors with last two digits of the serial number that are zeros (e.g. B8001600) are set to 100. No Logical address can be set to zero.

Enumeration vs. Override Enumeration Method

There are two methods to upload firmware code in a sensor. Enumerate and Override Enumeration. Override Enumeration method is used to detect a sensor specifying last three digits of the serial number. The method is used when two of the sensors in the system have the last three digits identical. It is a very unlikely yet possible. In such a case click the 'By Logical Address' box(Fig. 1.0) and type in the last 3 digits of the serial number (no need to type leading 0-s) then click 'Connect'. Follow procedure steps 6. and onwards to complete the upload.

5.4.NPH IP Address Setup

NPH concentrators are setup with a default IP address: **192.168.0.150**. However your networks IP address may be quite different and therefore the IP address must be permanently modified to become a part of your network sub-net. There are two ways to accomplish this:

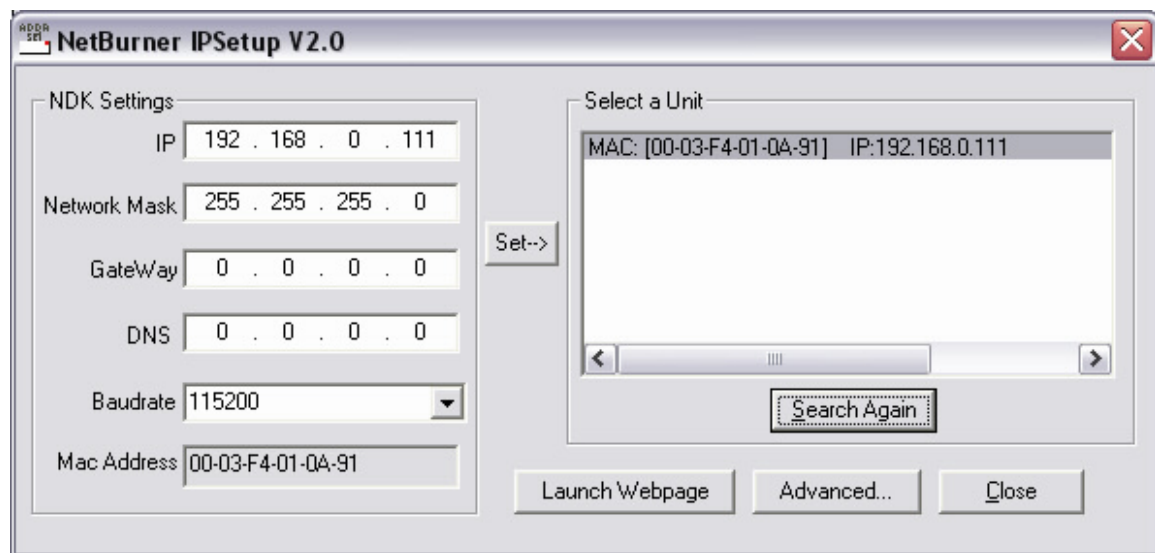
- Use **IPSetup.exe** application using network search for the NPH
- Use **MtTTY.exe** terminal program to permanently set the IP address over the serial link to the NPH.

Hardware

- NPH concentrator
- CAT5 Ethernet cable
- Serial communication cable 9-pin D connector
- Desktop PC / Notebook PC

5.4.1. NPH IP Setup Over the Network Procedure

1. Connect and power up both NPH and the client CPU.
2. Run the **IPSetup.exe** program. Following dialogue box will appear with the existing NPH IP address highlighted in the right dialogue pane:



IPSetup.exe

3. On the left side of the dialogue Enter the desired IP address and Network Mask on the left side of the dialogue and press set control button. The new IP address should appear on the right side of the dialogue, replacing the original, and default one.

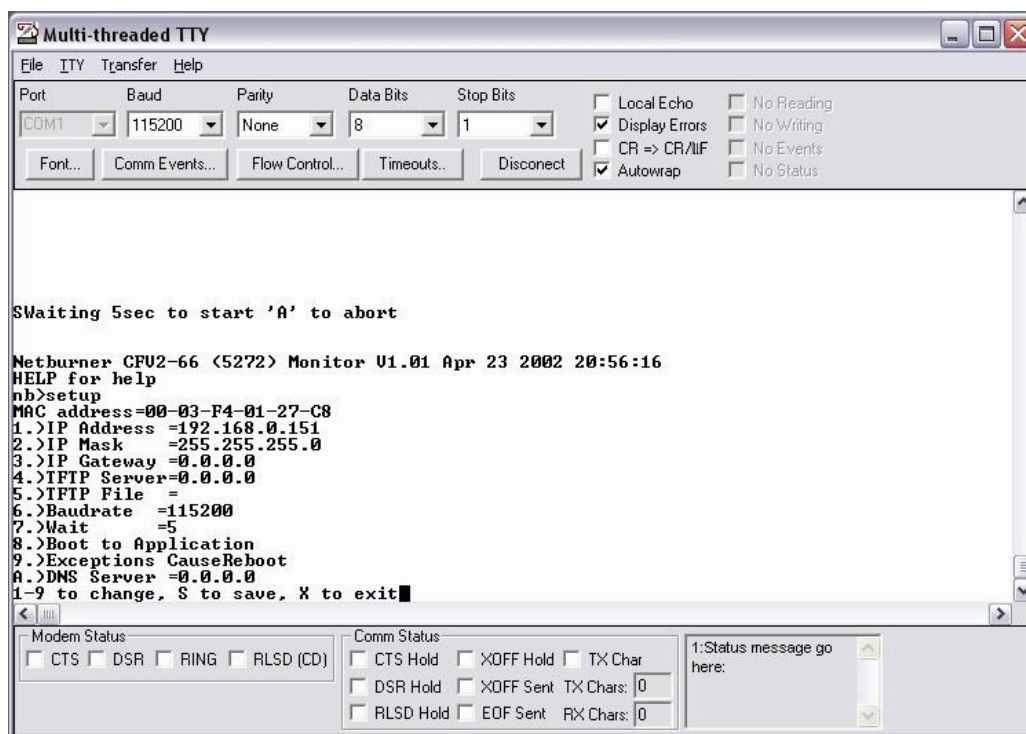
5.4.2. NPH IP Setup Over Serial COM Port Procedure

Using serial RS-232 connection

1. Connect COM serial port of the computer to the NPH.
2. Start MTTTY.EXE terminal program on the PC with following COM port settings:

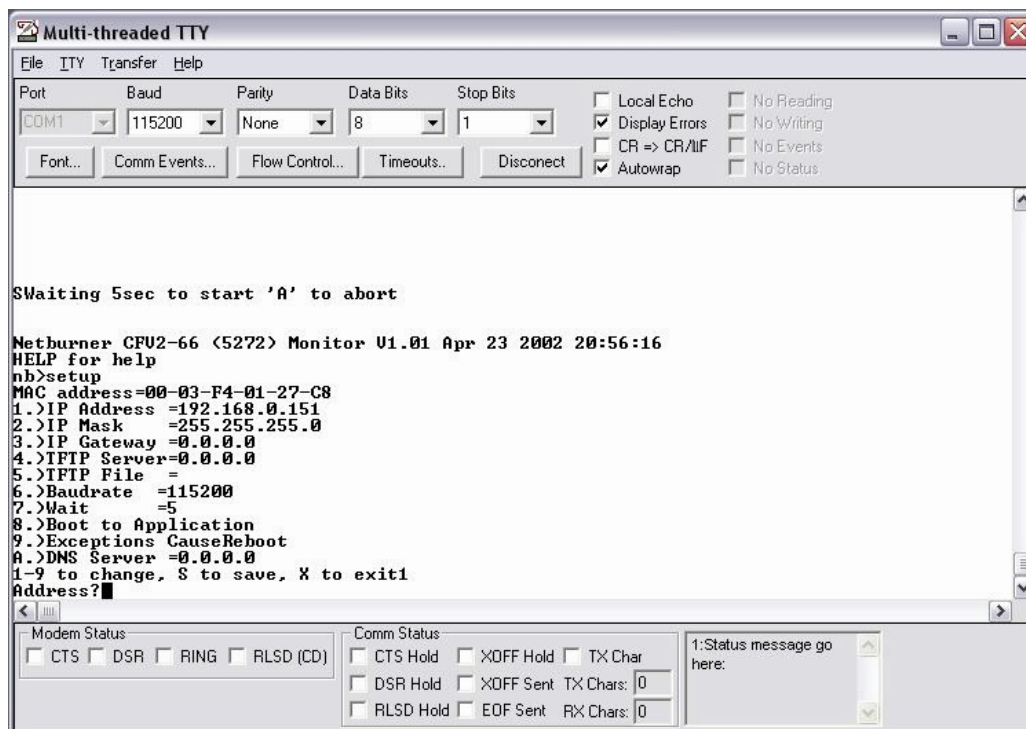
Baud	115,200
Parity	none
Data Bits	8
Stop Bits	1

3. Press **Connect** menu push button.
4. Power up NPH unit - red LED indicates ON state.
5. In the Mttty window a message will appear: **Waiting 5sec to start, 'A' to abort.** Press A (case sensitive) and a prompt will appear. At this point the system waits for any command.
6. Type **setup** at the command prompt and press **Enter**. Following message will appear:



MtTTY.exe

7. To edit new IP address type 1 and input your new IP address at the prompt:



MtTTY.exe

8. To accept the new IP address type S (save) and press Enter. The new IP address will be permanently programmed in and the NPH unit will go through the boot process, giving you the option to interrupt the boot-up process by pressing 'A' (case sensitive).

5.5. NPH Firmware Update

Following is the procedure to update NPH unit with a new code release. Firmware upload can be done either over the [RS-232 serial link](#) or over the [Ethernet connection](#). Since you are already connected to the NPH over the Ethernet it is more convenient and faster to update the NPH using existing network connection.

Hardware

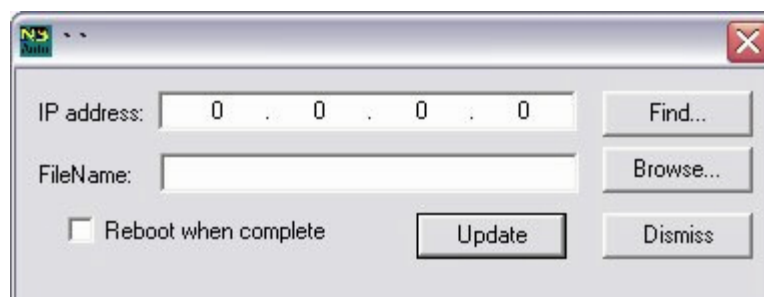
- NPH concentrator
- CAT5 Ethernet cable
- Serial communication cable 9-pin D connector
- PC / Notebook

Software

- MTTY.EXE
- *_APP.s19
- AutoUpdate.exe

5.5.1. NPH Upload Procedure using Ethernet

1. Connect and power up boot NPH and the client CPU.
2. Run the **AutoUpdate.exe** program.



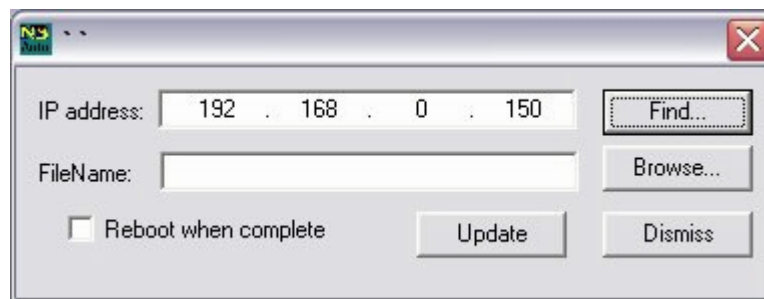
Mtty.exe, Main

3. Press *Find* button to detect the NPH unit on the network. Following dialog box will appear with the NPH IP address highlighted:



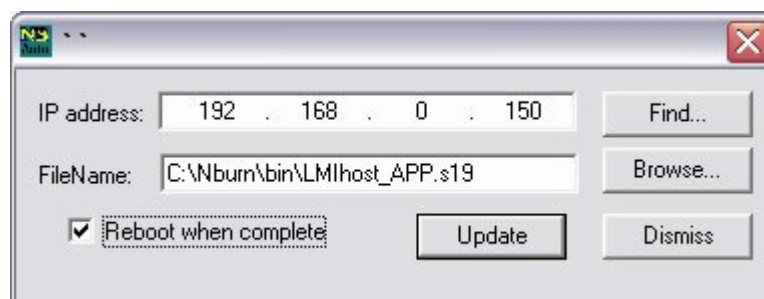
AutoUpdate.exe, Select NPH

4. Press *OK* to return. This time a correct NPH IP address should appear in the dialog *IP address* pane



AutoUpdate.exe, Main

5. Next press "Browse" button and that will open standard Windows File Open dialog. Find and select the **LMIhost_APP.s19** file supplied by LMI.
6. Check the *Reboot when complete* box and press *Update* button.



AutoUpdate.exe, Main

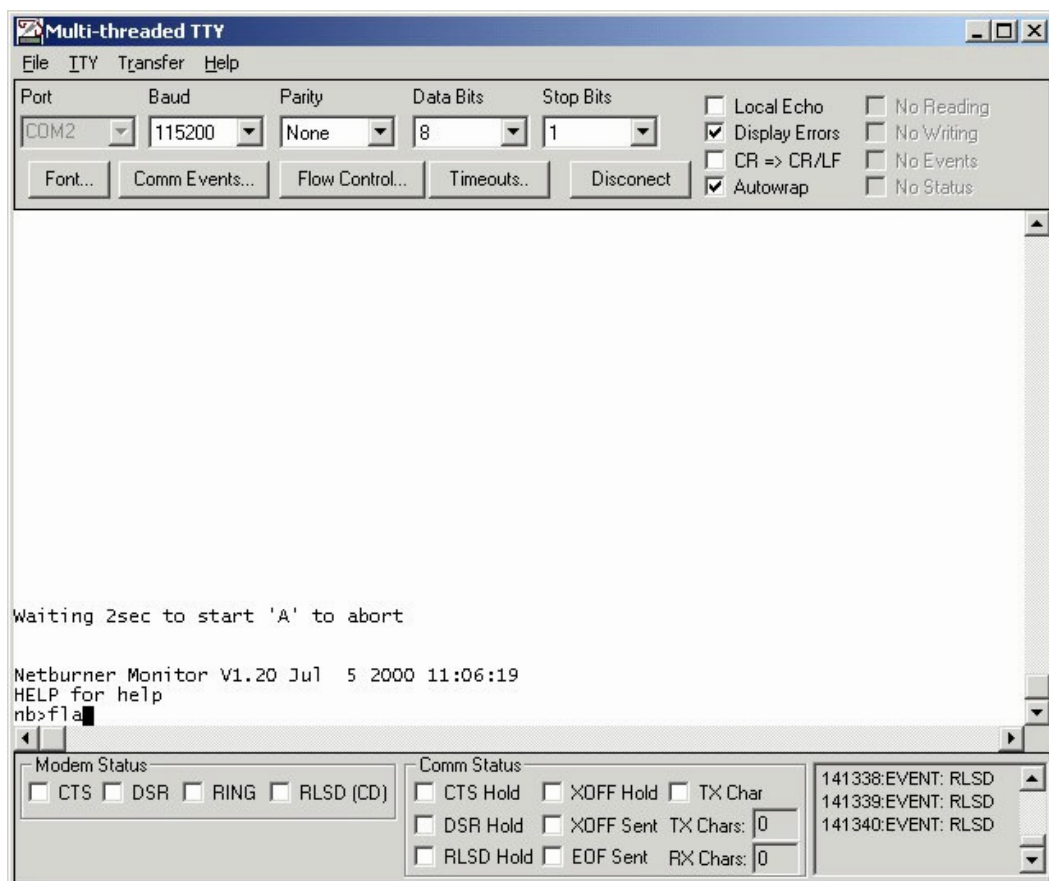
7. The program will permanently upload new code to the NPH unit. Upon completion of the upload the NPH will reboot and the new code will take effect.

5.5.2. NPH Upload Procedure using Serial Connection

1. Connect COM serial port on the computer to the NetPowerHub.
2. Start MTTTY.EXE terminal program on the PC with following COMM port settings:

Baud	115,200
Parity	None
Data Bits	8
Stop Bits	1

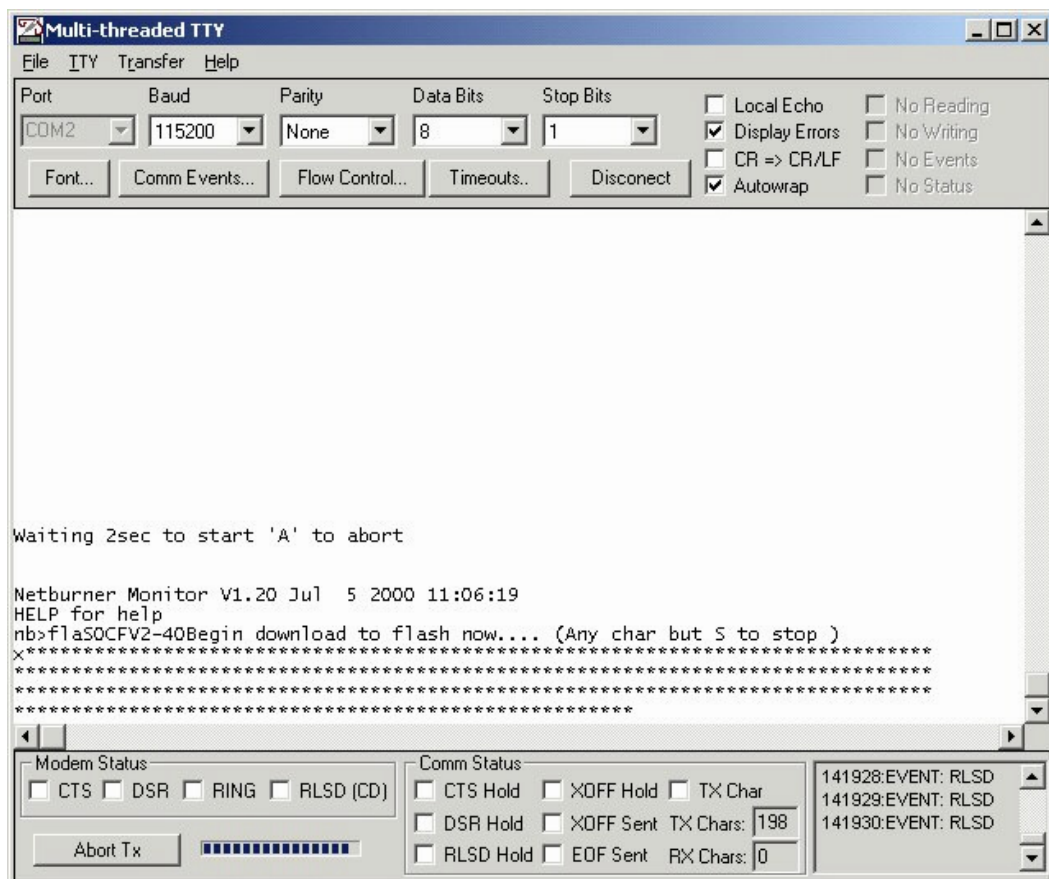
3. Press Connect menu push button.
4. Power up NPH unit - red LED indicates ON state.
5. In the Mttty window a message will appear: Waiting 5sec to start, 'A' to abort. Press A (case sensitive) and a prompt will appear. At this point the system waits for any command.



Mttty.exe , Begin serial FW upload

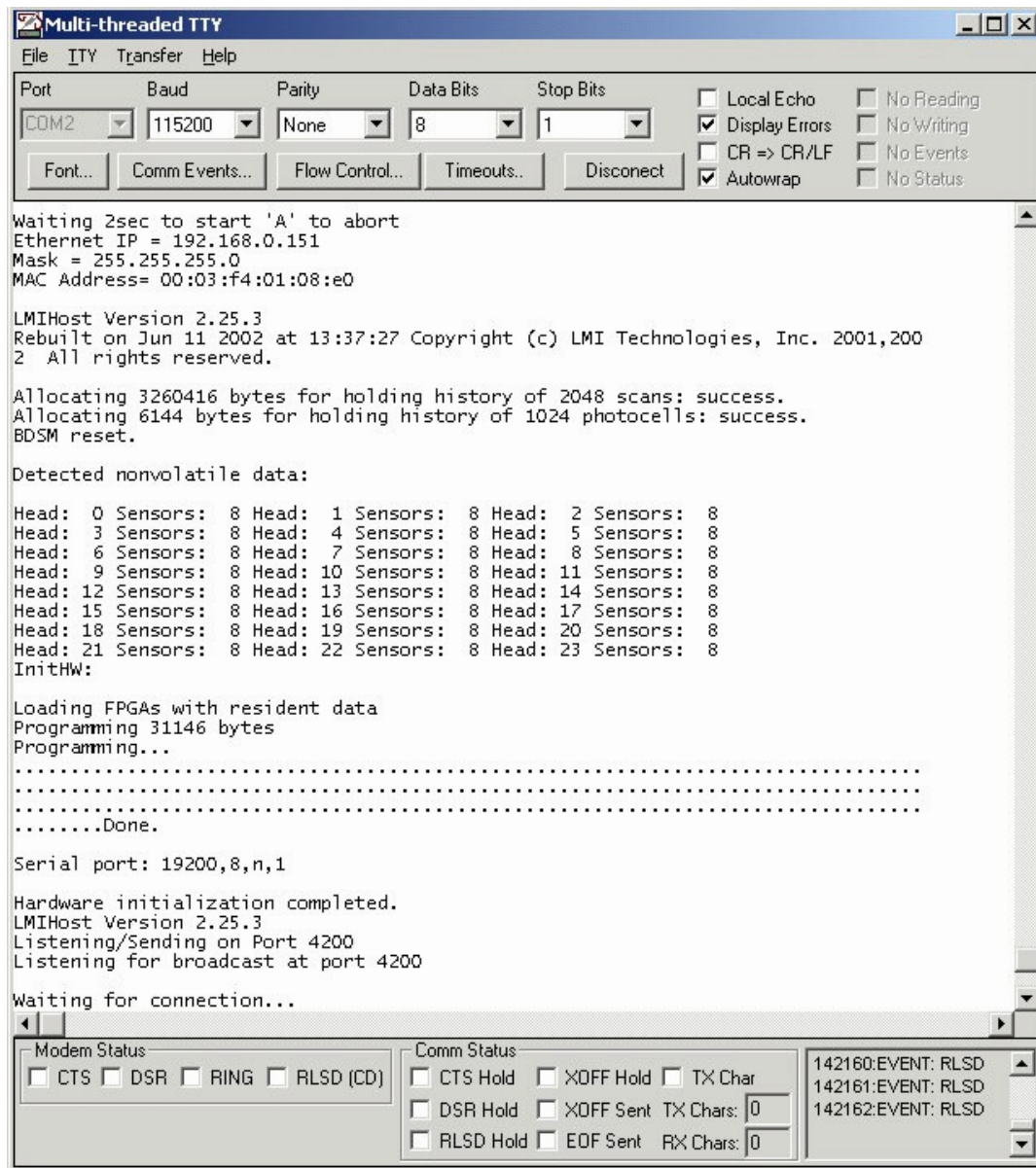
6. Type **fla** at the command prompt and press **Enter**. Following message will appear: **Begin download to flash now....** at this point select Menu item **Transfer** and from the submenu **Send** item. Or just hit F5 key. From open file dialog box select the name **_APP.s9** file

supplied by LMI. Once accepted, the file will be permanently programmed into the Flash memory of the NPH - **** indicates upload progress.



MtTTY.exe , Serial FW Upload

7. After the upload, every time the NPH is re-powered it will automatically boot up with this version of the code (always waits for few seconds). The upload can be done anytime so if you for example mistyped 'a' after power up just turn the unit OFF and ON again to try it again.



Mtty.exe, NPH Boot up

5.6. NbTest Diagnostics Program

NbTest.exe is a Win32 program running under Windows '9x/NT/2000/XP operating systems. The program is designed to verify correct setup and operation of a sensor based scanning system.

NbTest.exe uses following files:

nbllib.dll LMI API library

factor.dat A text file containing desired scanner parameters. This file can be edited/modified using any text editor to explore/test different scanner parameters/modes. If this file is **not present** the NbTest.exe will create one using default parameters once it is opened. The format of the file is one parameter/line (see [SCANNERPARAMS](#) structure).

offsets.dat This file contains system calibration individual lasers 'offsets' in encoder units in the direction of the target movement. This table can be produced using NbTest program. Initially the file if present will contain only 0-s for laser offset. User can run a straight edge board under the scanner (see [Scans Tab](#)) and once the board data are available pressing Recalc. button will use existing board data and calculate the system calibration values and store them in the Offsets.dat file for future use. These values can be cleared by pressing Clear push button.

Both *Factors.dat* and *Offsets.dat* are just an example format of preserving system data and used by NbTest.exe diagnostics program. User will have to implement their own method of storing scanner parameters initialization values and the system calibration data.

* some of the screens may differ from this documentation.

5.6.1. System Tab

Upon startup of the program the 'System' tab appears. Several functions can be executed pressing following buttons on the dialogue display:

Connect

Pressing Connect button will establish Ethernet communication with the NPH host concentrator. Nbtest application also receives the system status indicating presence of all operational sensors connected. The detected sensors will be indicated by a green square in the Heads array at the top of the dialog screen. Connection status is indicated in a window right of the Connect button, changing its background to green color and the "No Error" message.

Modify Logical Sensors Map

NPH concentrator is using a user defined logical table of connector vs. system sensor map thus allowing user to connect sensors to the NPH arbitrarily. This table is permanently stored in the NPH concentrator. NBTEST allows user to modify this table and permanently upload new table to the NPH (see [nbSetSensorMap](#) and

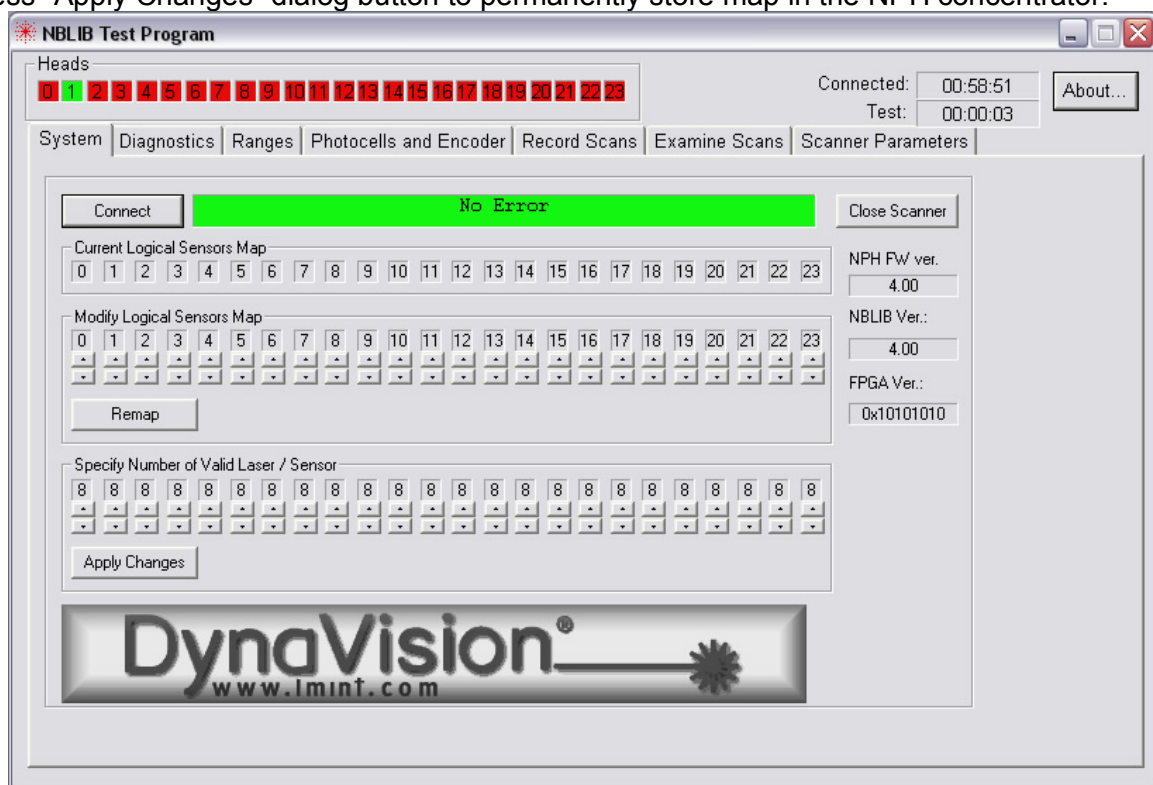
[nbGetSensorMap](#) API functions). Use the scroll controls to set sensor index vs. connector index and then press "Remap" dialog button to permanently upload the map to the NPH.

Specify Number of Valid Laser / Sensor

NPH concentrator is designed to accept inputs from different models of multipoint sensors. B8, B8A/Bx00, B16 and M24B. Using scroll controls set the appropriate value for each type sensor connected to respective connector.

B8A/Bx00 8
 B16 16
 M24B 23

Press "Apply Changes" dialog button to permanently store map in the NPH concentrator.



NbTest, System Tab

5.6.2. Diagnostics

Diagnostics Tab dialog window displays real-time data of a selected sensor. API call [nbGetLaserInfo\(...\)](#) and [nbGetHeadInfo\(...\)](#) are used.

- Ranges** Indicates real-time range reading for each laser beam
- Pwr** Indicates optimized Pulse Width Modulation (PWM) for given laser beam.
- Subpix** Location of detected laser spot image on each CCD camera in sub-pixel resolution.
- Sumpix** Integrated laser spot image area on each of the CCD cameras.
- Spots** Number of spot images detected on each CCD camera

Environmental diagnostics are displayed in the bottom right corner of the dialog window. Packet indicator indicates number of correct TCP/IP packets received by the client.

NBLIB Test Program

Heads: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Connected: 00:59:21
 Test: 00:00:02

System | **Diagnostics** | Ranges | Photocells and Encoder | Record Scans | Examine Scans | Scanner Parameters

	RANGES	PWR	SUBPIX		SUMPIX		WIDTH		SPOTS	
Spot 0	20039	33	10690	14589	2112	677	28	8	8	8
Spot 1	20006	25	17468	18291	1999	573	17	6	8	8
Spot 2	19970	27	24140	22624	1852	709	12	8	8	8
Spot 3	19927	19	30440	27617	1133	674	12	9	8	8
Spot 4	19907	16	36179	33325	644	582	10	9	8	8
Spot 5	19875	23	41251	39610	807	1413	9	11	8	8
Spot 6	19838	16	45676	46352	769	1525	7	14	8	8
Spot 7	19817	24	49466	53238	590	1316	7	21	8	8
Spot 8										
Spot 9										
Spot 10										
Spot 11										
Spot 12										
Spot 13										
Spot 14										
Spot 15										

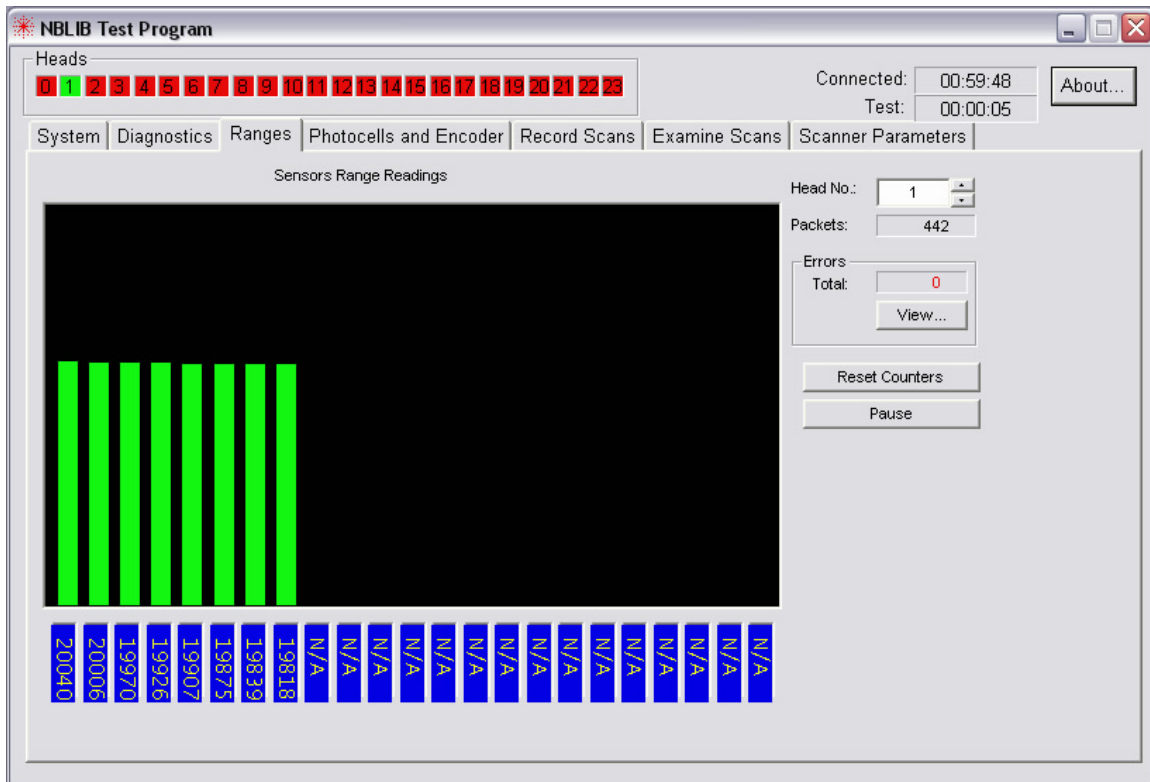
Head: 1
 Packets: 108
 Errors Total: 0
 View...
 Reset Counters
 Pause
 Save Laser Diags

S.N. #: 6001946
 Model: 8005
 DSP Rev.: 112
 FPGA Rev.: 111
 Bus Addr.: 46
 Temp (C): 33.00
 VCD (Volt): 18.90

NbTest, Diagnostics Tab

5.6.3. Ranges

Ranges Tab windows displays both numerical and a bar graph representation of current range readings. API call [nbGetRanges\(...\)](#) is used.

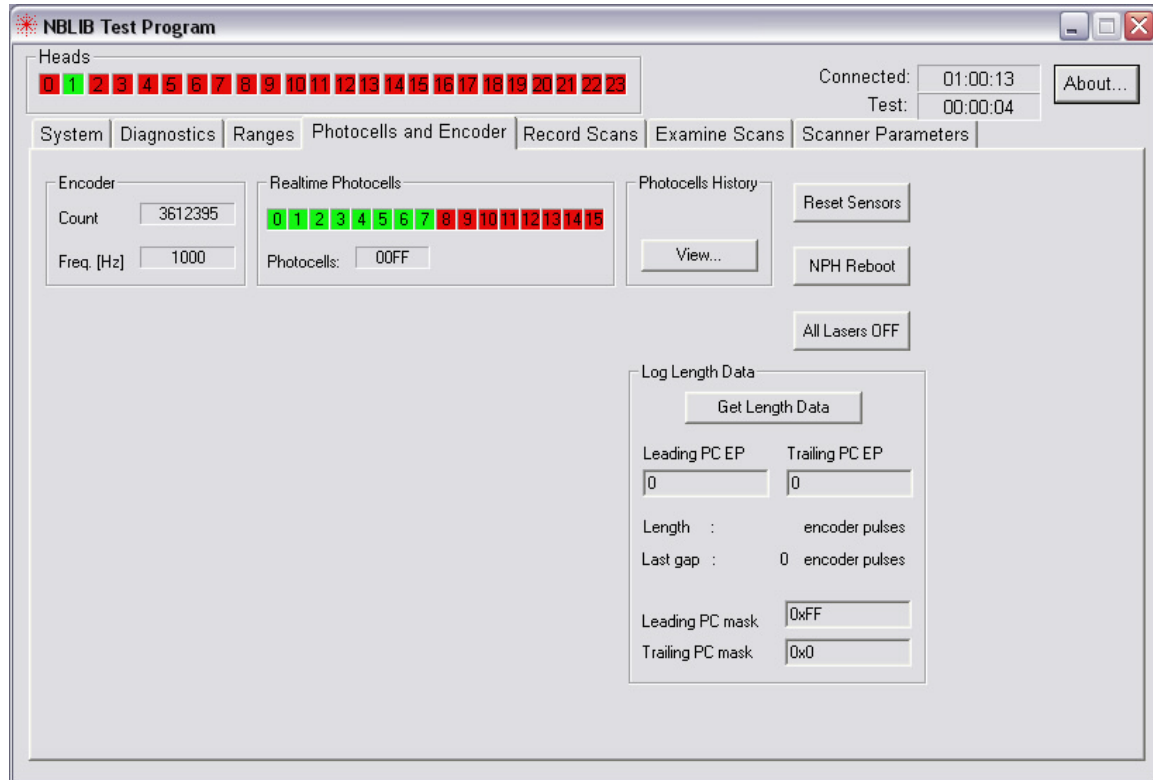


NbTest, Diagnostics Tab

5.6.4. Photocells and Encoder Tab

This dialog window displays real-time encoder and photocell status. Current NPH supports up to 8 photocells input. Open (light) state of a photocell is indicated by a green square. Blocked (dark) photocells are indicated by a red square.

Photocell History - View button uses [nbGetPhotocellsHistory\(...\)](#) API call and will display retrieved data a text format.



NbTest, Photocells and Encoder Tab

5.6.5. Record Scans Tab

This Tab dialog shows "bird's view" scanned object data. Left pane displays real-time board data collected on each encoder pulse up to either maximum allocated board buffer space or until the target (board) left the scanner. Each horizontal line represents one scanline - collection of range data from all installed sensors at given encoder pulse. Green color denotes range readings within the calibrated range and yellow color represents out of range data (0x8000).

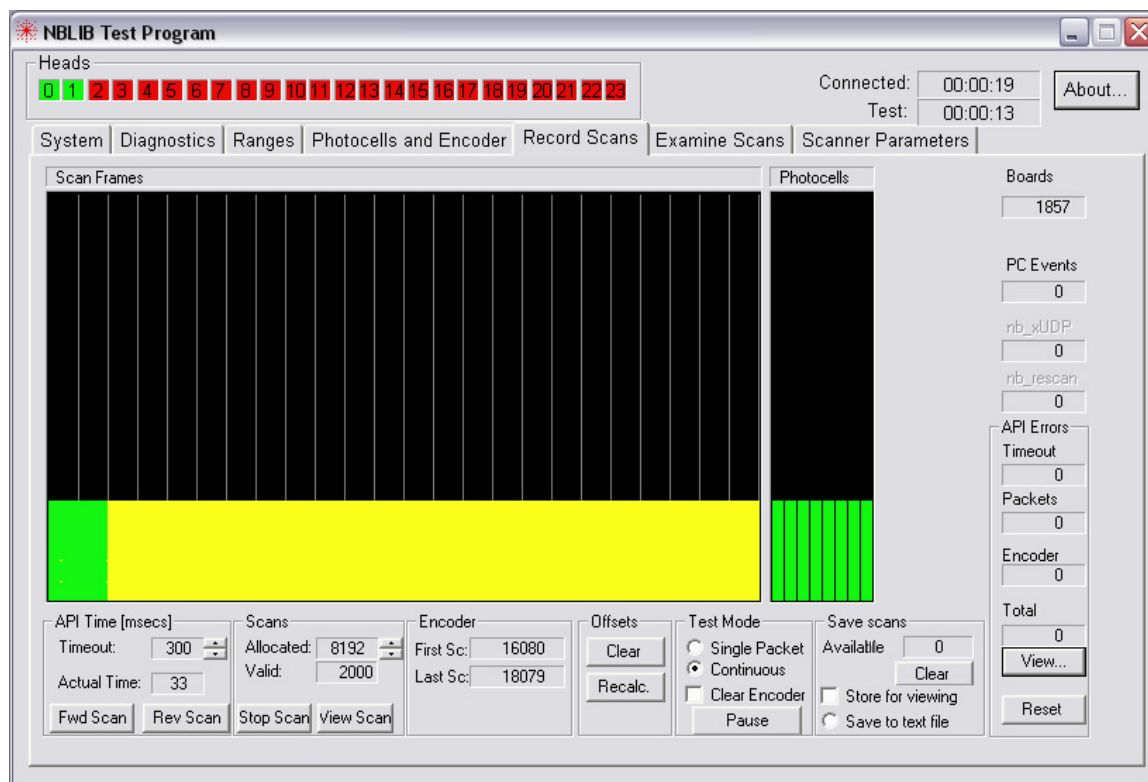
Right pane displays related photocell status. Green color represents open state of a photocell and yellow blocked state.

- "View Scans" button enables user to view board buffer data in formatted text format.

- "Save Boards to File" check box automatically saves all scans to new file labeled with the board number.

- "Store for viewing" check box stored scans for viewing on the "Examine Scans" tab.

- "Fwd Scan" & "Rev Scan" buttons are used for scanning in mode 4 depending on encoder direction.



NbTest, Scans Tab

5.6.6. Examine Scans Tab

This dialog window displays all scans that you have stored for viewing from the previous “Examine Scans” tab. The maximum number of scans that can be stored for viewing at one time is 200.

-“>” “>>” & “<<” “<” buttons will allow you to toggle through the saved scans.

-“Clear All” will clear all the scans that have been stored for viewing.

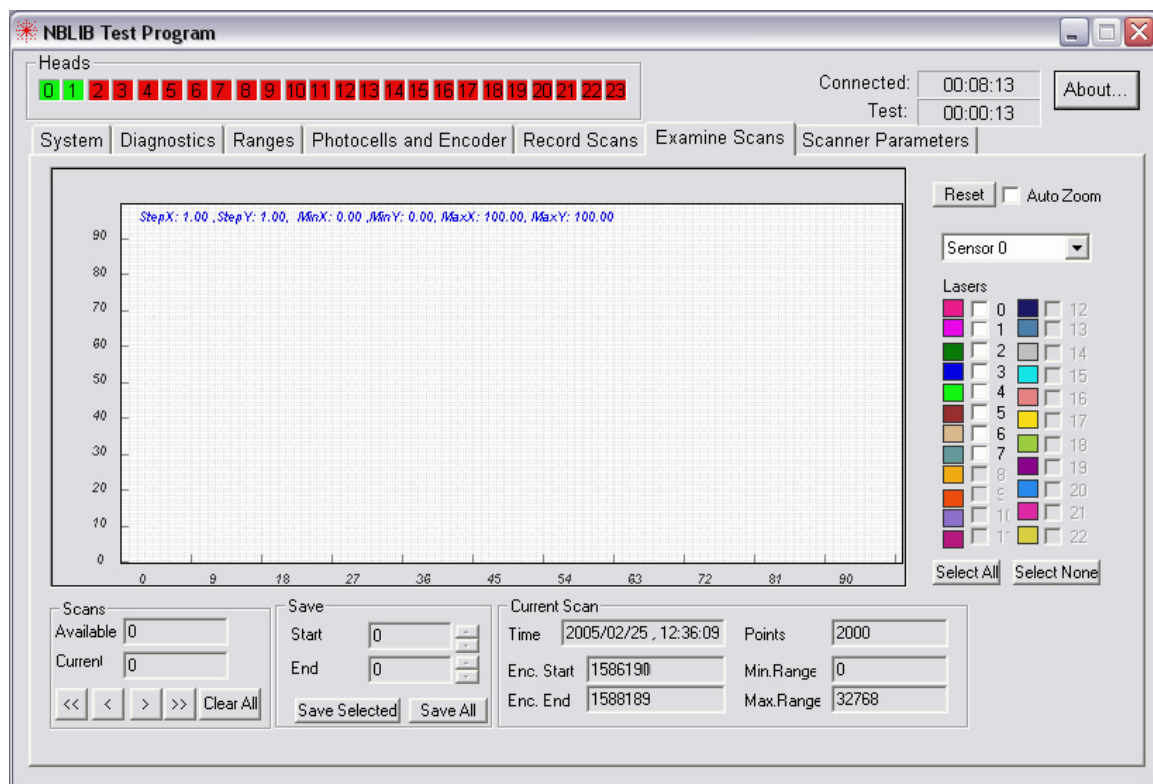
-“Save Selected” will save all the scans between the range indicated by the “Start” and “End” fields.

-“Save All” will save all the scans that have been stored for viewing.

-“Auto Zoom” check box will zoom into the current scan automatically.

-“Reset” will reset the scan view to the original view.

-“Select All” & “Select None” will select and deselect all lasers displayed on the graph. Individual lasers can also be selected by selecting the appropriate checkbox.



NbTest, Scans Tab

Note: You can zoom in on the graph by using your mouse to select the region to zoom into.

Scanner Parameters Tab

This dialog window displays all scanner parameters that the NPH has booted up with. These parameters can also be found in the Factor.dat file. On this page parameters can be modified as required. When changes are all complete the “Accept New Parameters” button should be selected, and **parameters will take effect once you have disconnected and reconnected to the NPH**. The NPH IP address can also be selected so that nothing else is disconnected on your network. If no IP is entered (IP is 0.0.0.0) then the NPH will search for your NPH and connect to the first one that it finds. Note that if there are multiple NPH concentrators on your network it is recommended that you enter the IP of your NPH to avoid conflict. Also if there are multiple Net Hubs online you should connect via crossover cable so no other client pc will try to connect to your NPH while you are running.

The screenshot shows the 'NBLIB Test Program' window with the 'Scanner Parameters' tab selected. At the top, there is a 'Heads' section with a row of 23 colored squares (0-22). Below this is a tabbed interface with 'System', 'Diagnostics', 'Ranges', 'Photocells and Encoder', 'Record Scans', 'Examine Scans', and 'Scanner Parameters'. The 'Scanner Parameters' tab contains three main sections:

- Scanner Parameters:** A list of parameters with input fields: Lead spots (5), Lead wait (3), Trail Spots (5), Trail wait (3), Lead History (0), Maxwidth (500), Max reverse (50), Mode (0), Trigger index (0), Encoder (0), Length index (1), Segments (1), Trail History (0), StopScan Index (2), and Timer Frequency (0).
- NPH-66 IP Address:** Four input fields showing '193', '168', '0', and '1111', with 'Accept New Parameters' and 'Cancel' buttons below.
- Serial Port Settings:** A box containing 'Baud Rate' (115200), 'Data Bits' (8), 'Parity' (none), and 'Stop Bits' (1), with an 'Apply' button.

At the top right, there are 'Connected:' (00:03:32) and 'Test:' (00:00:16) timers, and an 'About...' button.

NbTest, Scanner Parameters Tab

6. Getting help

If you wish further help on the component or product contact your distributor or LMI directly. Visit our website at www.sensorsthatsee.com for the agent nearest you.

For more information on Safety and Laser classifications, contact:

Center for Devices and Radiological Health, FDA
Office of Compliance (HFZ-305)
Attn: Electronic Product Reports
2098 Gaither Road
Rockville, Maryland 20850