USER'S MANUAL

MB Station

Version 5.1.6.0

Proprietary

Trademarks and Restrictions

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.

LMI Technologies, Inc.
1673 Cliveden Ave.
Delta, BC V3M 6V5
Canada

Telephone: +1 604 636 1011
Facsimile: +1 604 516 8368
www.lmi3D.com

Table of Contents

Section 1

1 Introduction

The MB Station is the next generation communication interface for all DynaVision® multipoint sensors.

The MB Station makes upgrade from NPH66 extremely easy. It uses the same cable connections, and bolts directly in place of the NPH66. MB Station incorporates hot swapping protection and will not be damaged if any of the connectors were inserted or removed while the power is turned on.

The field-wireable terminal strips provide easy and secure connection for power, encoder and photocell inputs. The Phoenix terminals on the MB Station are well suited for easier access and faster connection. A standard RJ45 connection provides Ethernet out.

The design incorporates a number of LED’s which help monitor the activity on your MB Station. The connector spacing on the face of this unit has been altered to make it easier to plug and unplug sensors.

Existing software written for the NPH-66 should not require any changes to work with MB Station. Full quadrature division support has also been incorporated into the B-series software and is available with MB Station.

2 MB Station Description

The MB Station allows for up to 24 sensors to be connected via high speed serial lines, while providing each sensor with individually fused DC power. The MB Station includes Encoder Input with LED signal indicators and maximum 6 Photocell Input with LED indicators.

The MB Station supports all B-series sensors including the M24B.

- Full compatibility with existing NPH-66 external connections and cabling (sensors, photocells, encoder)
- Full compatibility with NPH-66 Ethernet protocol and existing software suite
- Hot swapping protection prevents damage to the MB Station.
- 2 kHz data sampling and streaming is supported for full B-series and M24B sensor systems
- Encoder quadrature division is supported

Figure 2.1 –MB Station

2.1 MB Station Features

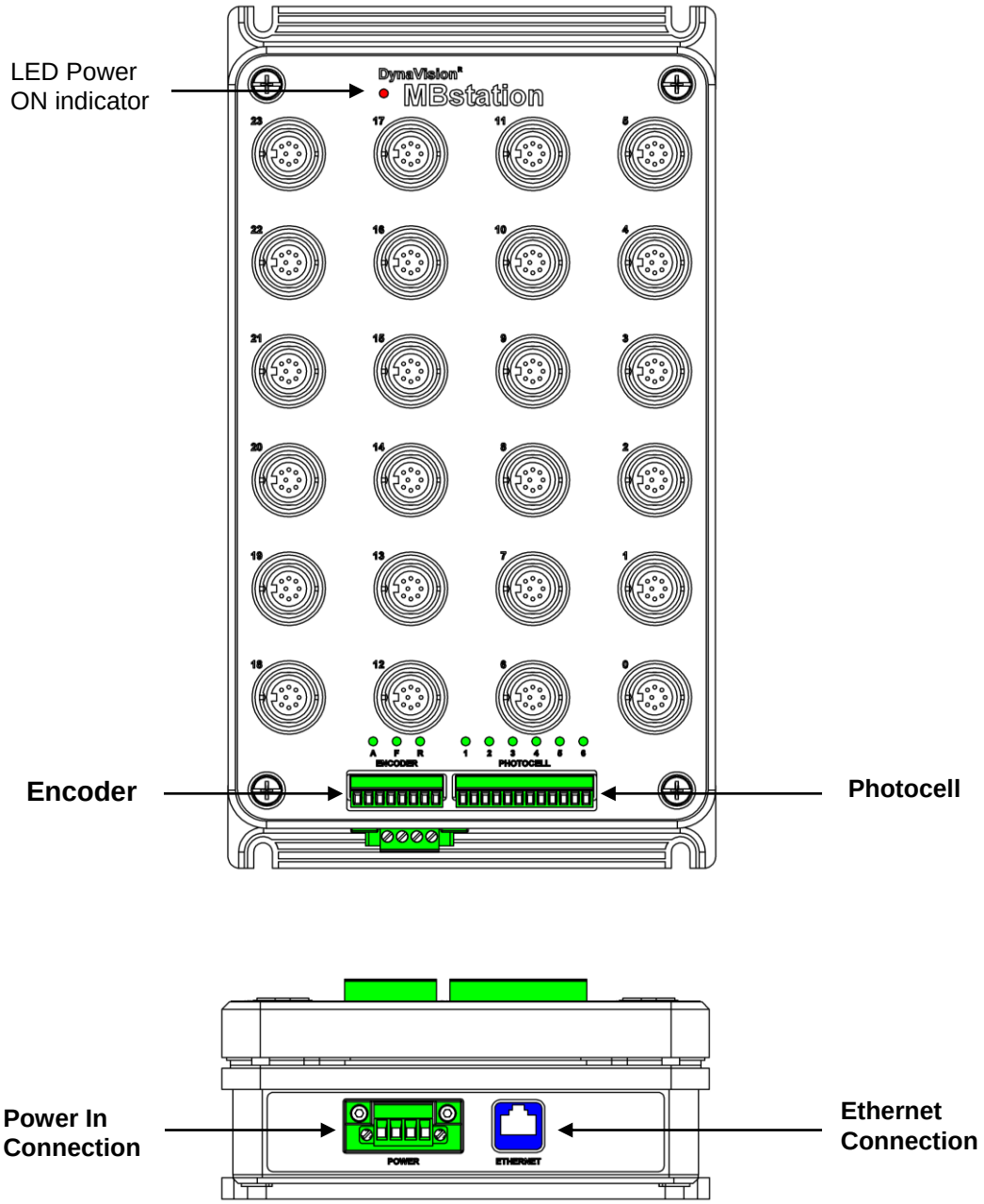


Figure 2.2 – Front and Bottom Layout of the MB Station

2.2 Storage Considerations

The MB Station must be mounted inside of an industrial NEMA enclosure at the scanner frame to protect the electronics. 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 cabinets are not positioned where they are in direct view of any of the sensor cameras.

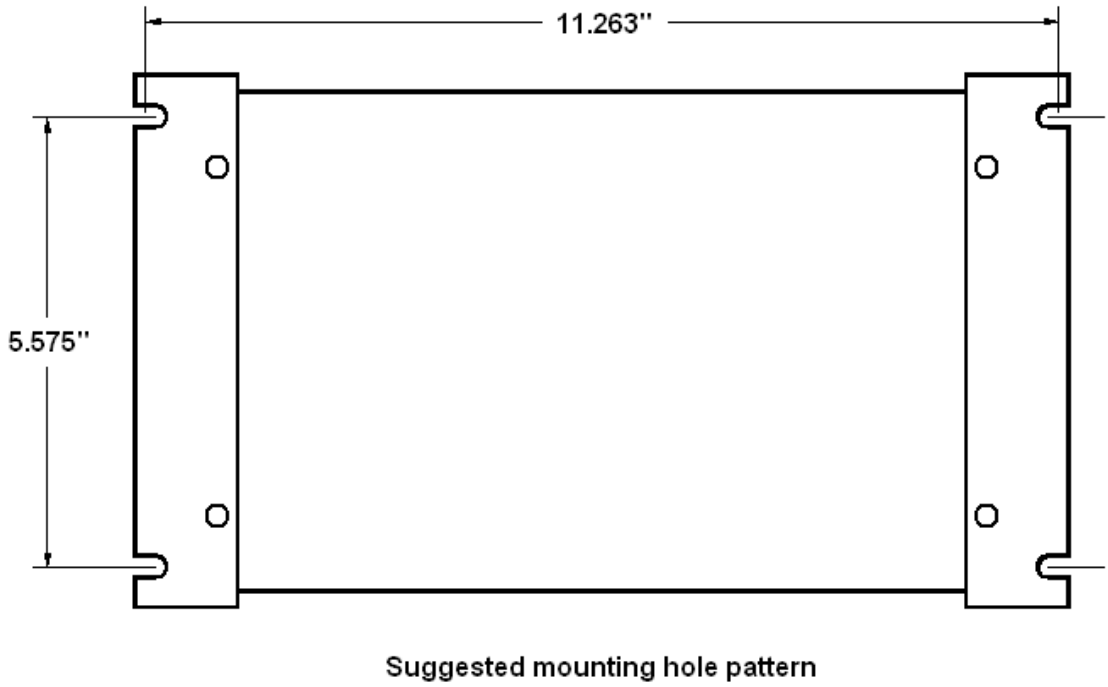


Figure 2.3 – Suggested mounting hole layout of the MB Station.

3 Connecting the Hardware

Always make sure that the power to the MB Station is off and that all cables are pre-tested before connecting to the unit. *Note that most common problems occur in the cable connection between the sensor and the MB Station unit. Always double check your cables for shorts or faulty wiring.* Please refer to the Connection Diagram below.

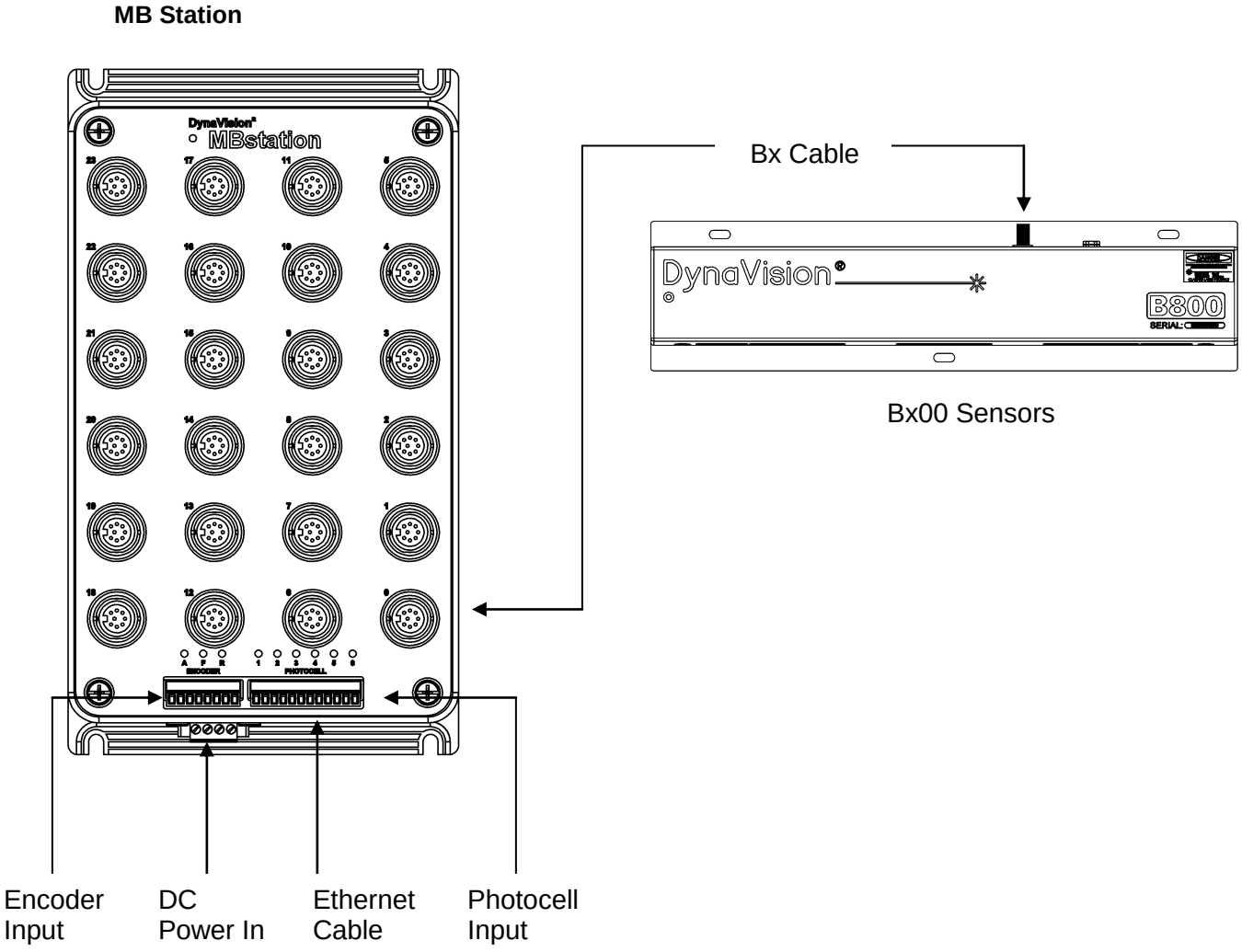
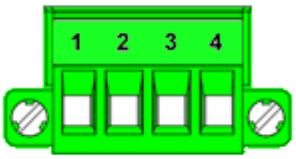


Figure 3.1 – Connection Overview

3.1 DC Power Input

Please verify the DC power cables for correct pin outs before connecting to the MB Station. **WARNING**: Neither the MB Station nor the sensors include reverse and over voltage protection. When the MB Station is powered on the green LED power indicator will light on.

Pin	Assignment
1	+24VDC
2	+24VDC
3	GND
4	GND



www.phoenixcontact.com
P/N 1786578 MSTB 2,5/ 4-STF

Figure 3.2 – DC Power Input pin out

3.2 Connection Overview

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 MB Station. Position is not significant (any cable can go to any available connector). Suggested bolt size: #8 to #10

3.3 Connecting Sensors

When connecting sensors please ensure that the cables are fully pushed into place and that the hold down sleeve is screwed on hand tight.

3.3.1 Sensor [Input/Output]

MB Station provides DC power and proprietary high speed differential serial link of up to 24 installed sensors. Power supplies to operate the sensors should be mounted in a cabinet on or near the sensor frame to reduce voltage drops to each sensor.

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

Cable:
The power cable connector used is an **Amphenol, C091 31H008 101 2**

3.4 Bx Cable Specification

3.4.1 Bx Power/Data Cable

A single four pair plenum grade (i.e. industrial) shielded CAT5 cable is run to each head from the MB Station. The power cable connector used is a DIN 41 326 Standard 8 pin circular connector. Pin outs are below as viewed from the solder side:

PIN	Assignment
1	RxTx **(+)**
2	SerialClock **(-)**
3	SerialData **(-)**
4	VDC **(+)**
5	SerialClock **(+)**
6	RxTx **(-)**
7	SerialData **(+)**
8	VDC **(-)**

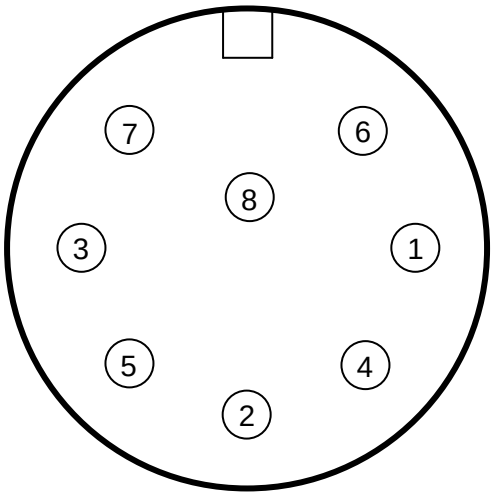


Figure 3.3 – Pin out of the cable from the solder side (inside).

Name	Description
VDC+/-	DC power (VDC+) and Ground return (VDC-)
RxTx+/-	Asynchronous 19.4 kBaud serial bi-directional pair (RS-485)
SerClk+/-	High speed Serial Clock pair
SerDat+/-	High Speed Serial Data pair

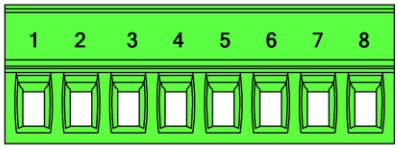
3.5 Connecting the Encoder

The encoder should be installed to be driven from the sensor transfer chain. The encoder is wired to the computer cabinet with a shielded cable in separate conduit from the encoder to the MB Station unit via Phoenix connector.

3.5.1 Encoder Input

MB Station supports differential quadrature opto-mechanical encoder modules. Maximum recommended resolution is 2 kHz. The MB Station 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. Pull the Encoder connector (8 pin Phoenix, MCVW 1,5/ 8-ST-3,8, 1827033) and wire as specified below. Snap connector back in.

PIN	Assignment
1	A+
2	A-
3	B+
4	B-
5	Z+
6	Z-
7	GND
8	+5V



www.phoenixcontact.com
P/N 1827033 MCVW 1,5/ 8-ST-3,81

Figure 3.4 – Encoder connector pin out

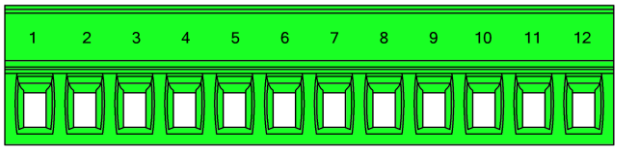
3.6 Connecting Photocells

The MB Station accepts up to 6 opto-coupled photocells via Phoenix connector on the face of the unit.

3.6.1 Photocell Input

Photocells generally have an open-collector transistor output that operates as an on/off switch. Each photocell input on the MB Station has to have a corresponding external pull-up resistor implemented by the customer that allows the photocell switch to generate **High** and **Low** states in the MB Station (see the recommended schematic **3.6**). Pull the Photocell connector out (12 pin phoenix Phoenix, MCVW 1,5/12-ST-3, 81827075). Wire as specified below. Snap connector back in.

Pin	Signal	Photocell ID
1	+IN	1
2	-IN	
3	-IN	2
4	+IN	
5	+IN	3
6	-IN	
7	-IN	4
8	+IN	
9	+IN	5
10	-IN	
11	-IN	6
12	+IN	



www.phoenixcontact.com
P/N 1827075 MCVW 1,5/12-ST-3,81

NOTE: The GND pin on the 8-Pin Phoenix connector (Encoder) can also to be used as GND for the photocell input.

Figure 3.5 – Photocell connector pin out

The following diagram shows the connection of the photocell to the MB Station circuitry:

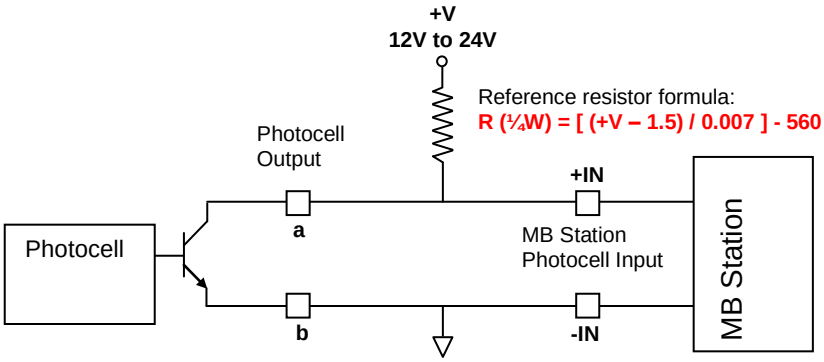


Figure 3.6 – Photocell connection schematic

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 MB Station to ground.

*Photocell input **pins a – b** correspond to the photocell **inputs +IN … -IN**. The state of any unused photocell inputs doesn't matter (They can be grounded).

4 Scanning System

B-series systems utilize the MB Station in many different applications. In addition to transverse board scanning, it supports longitudinal and rotational log scanning. MB Station also supports a software triggered mode and a continuous (pass through) mode, which can be applied to almost any situation.

4.1 Scanner System Components

Following scanner components are supplied by LMI Technologies:

- Multipoint range measurement sensor(s).
- MB Station concentrator unit.
- Integrated data / power Bx Cable.

The OEM must supply the following components:

- Scanner frame to mount the sensors.
- 24V DC power supply to power the sensors (15-24 watts each)
- Power supply cabinet to host the DC power supply and the MB Station 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 MB Station Phoenix 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.
- A 2” by 4” by “board length” profile bar to calibrate the BX system.

4.2 Scanning modes

4.2.1 Board Detection Mode (0)

Board detection mode is typically used for transverse board or cant scanning. The overall process is as follows:

The start of a board is detected whenever a **lead_spots** (see scanner parameters structure) or greater number of valid ranges is present for **lead_wait** number of scans. At this point MB Station starts sending scanned data to the client. The actual start of the sent data is **history** scans before the start of the board. Each scan includes sensor data for the entire system as well as encoder and photocell/input state at that point. The end of the board is triggered when **maxwidth** number of scans is reached or fewer than **trail_spots** ranges are valid for **trail_wait** number of scans. Board data is transmitted until additional **trail_holdscan** number of scans has been sent or **maxwidth** number of scans has been reached. When all data for a given board has been received by the client library, a Board Ready Event is issued.

Scan acquisition and transmission are independent, and have different timing requirements. MB Station only stores information about the last scanned object. This means that if more than one object has been scanned in addition to the one currently being sent, the last one of the new objects will be sent and the others will be lost. Correctly choosing board spacing and **lead_spots/lead_wait** parameters is required for correct operation.

For example, if **history** were set to 5, **lead_wait** set to 3, **trail_wait** set to 3, and **trail_holdscan** set to 5, and if the scanned board lasted 250 frames before the first of **trail_wait** (3) frames that did not satisfy **trail_spots**, the returned board length would be calculated as follows:

5 + 3 + 250 + 3 + 5 = 266 frames

4.2.2 Longitudinal Log Scanning Mode (1)

This scanning mode requires an external trigger, a photocell or a switch connected to one of the 6 available photocell inputs on the MB Station 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 MB Station unit to start buffering real-time range data from all the installed B-Series sensors at each **forward** encoder pulse.

To optimize, the time data is 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), **nblib.dll** will issue a *Board Ready Event* to the user application. 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 B-Series sensors detect the target anymore and the photocell used for the trigger is in the *light* state. The actual end of log is located

somewhere within the last scan segment and therefore the user software must locate the last valid data to completely define the log model. Log end detection logic is similar to **Mode 0** as it also uses **trail_wait** and **trail_spots** parameters. However, trailing history is not supported in this mode. The length of the log can also be determined by the number of encoder ticks counted while the photocell is dark.

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 is being transmitted over the network.

4.2.3 Rotational Log Scanning Mode (2)

In this case a log is positioned under the scanner on a spindle. The 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 processing, the user can specify that the system should segment the data. In the case above, the user could specify to retreive data in segments of 90 measurements. Thus the whole scan will be delivered in 4 segments, each 90 data frames wide.

Board Ready Event is issued after each segment has been received. 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.

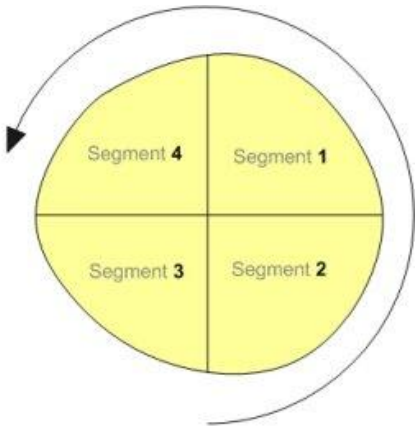


Figure 4.1 – Example of how a 360 degree measurement can be specified into four, 90 degree segments

4.2.4 Continuous Scanning 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 data frames on each encoder pulse, forward or reverse. **nblib.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 transfers the section data into its own data space and executes data processing. Meanwhile another data section is being collected.

4.2.5 Software Triggered Mode (4)

This mode is very similar to **Mode 2**. In the Software Triggered Mode, the triggers to start and stop scanning come from the client application. See *nbStartScan()* and *nbStopScan()* for more details. After the **start command** is issued, the MB Station continuously sends scanned data to the client delimiting each segment at **maxwidth** interval. This continues until the number of segments reaches the value of the **segments** parameter, or the **stop command** is issued. After completing this process, the MB Station waits for another **start command.**

4.3 Log Length Information

On each photocell transition, light to dark or dark to light, the MB Station records the 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, the 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, the user is able to calculate current length of the log accurately. Second, the user has information on the 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 MB Station using the TCP/IP protocol. See *nbReadLogLength()* API call for more explanation.

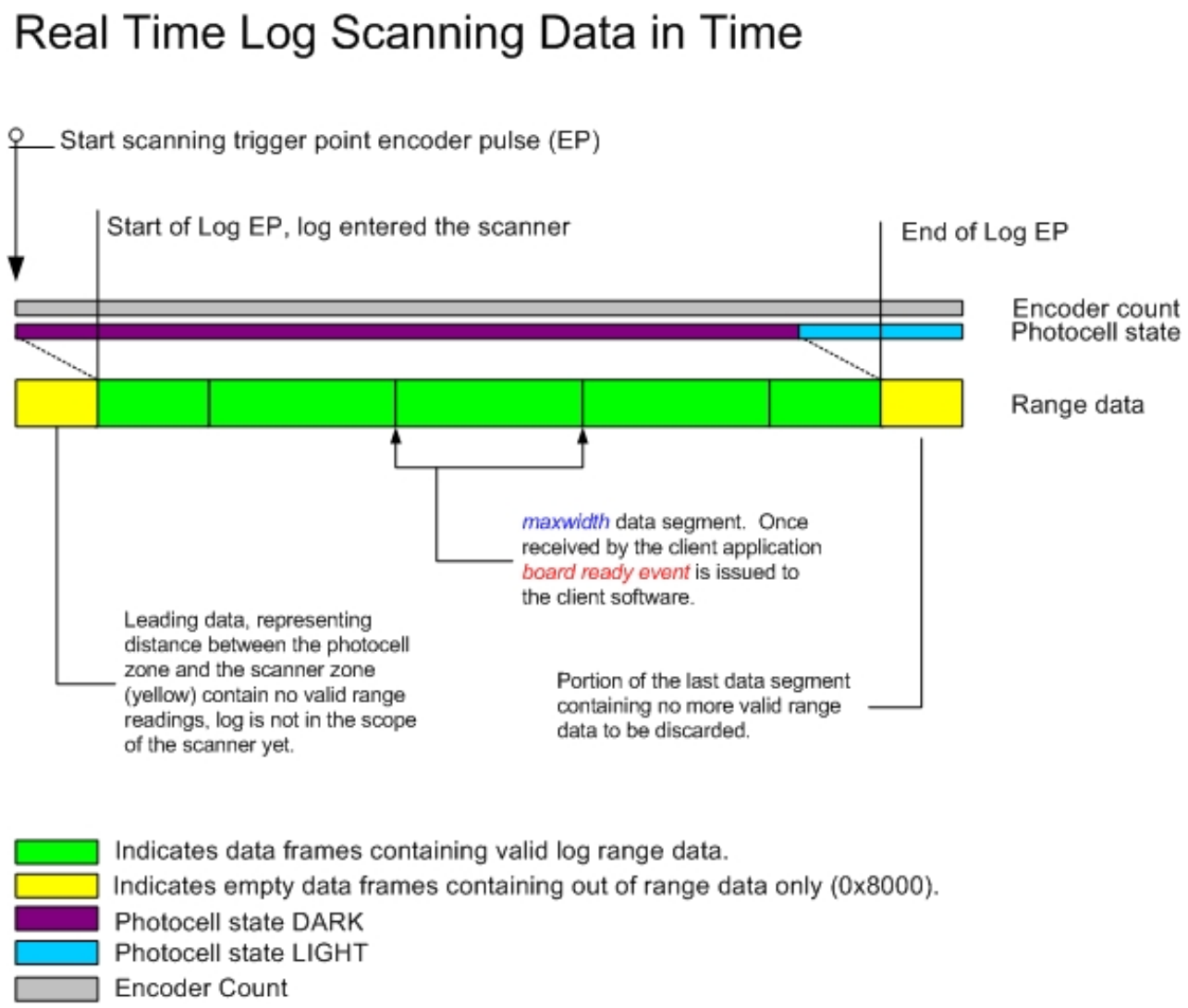


Figure 4.2 – Diagram of Real Time Log Scanning Data

5 API Functions

The functions described in this section provide programming interface between the client computer and the MB Station using NBLIB.DLL LMI API library. Communication uses TCP/IP and UDP protocols over the Ethernet network connection between the MB Station 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.

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

5.1 NbLib API Functions

Table 5.1 – NbLib API Functions

nbOpenScanner	Initializes the ethernet interface.
nbCloseScanner	Terminates the ethernet interface.
nbGetSystemDataInfo	Returns information about data capabilities of the system.
nbGetLaserStatus	Returns individual laser status (disabled or enabled).
nbSetLaserStatus	Disable/enable laser readings.
nbGetLaserInfo	Returns diagnostic information from a sensor.
nbGetHeadInfo	Returns individual sensor diagnostics.
nbGetEncoderInfo	Returns the current encoder count.
nbSetEncoderInfo	Sets encoder counter.
nbGetPhotocells	Retrieves real-time photocells status.
nbGetPhotocellHistory	Retrieves photocell history buffer.
nbGetScannedObject	Detects and collects scanned object / board.
nbGetScannedObject2	Detects and collects scanned object. Provides light curtain output.
nbGetRanges	Returns real-time range readings.
nbGetAllRanges	Returns real-time range readings from all the installed sensors.
nbGetVersion	Returns API library software version.
nbGetFwVersion	Returns MB Station code version.

nbGetFPGAVersion	Returns FPGA code version.
nbGetLastAsyncEvent	Returns the last asynchronous event information.
nbGetOffsets	Read current system calibration data.
nbSetOffsets	Set system calibration data.
nbGetSensorMap	Returns a mask of attached sensors.
nbSetSensorMap	Set the map of attached sensors.
nbGetLastError	Retrieve last API error.
nbGetValidLasers	Read number of valid lasers a given sensor is using.
nbSensorReset	Reset an individual sensor.
nbUploadUserParameters	Upload user parameters to the MB Station concentrator.
nbGetSystemData	Returns diagnostics for all sensors in the system
nbReadLogLength	Upon ASYNC_EVENT_LOG_PC_INFO event, read given photocell transitions record.
nbResetNPH	Client command to reset/reboot MB Station unit. **Currently not supported by the MB Station.**
nbSerialSettings	Initialize MB Station COM port.
nbStartScan	Software trigger of scan frames acquisition upon each encoder pulse.
nbStopScan	Abort/complete data acquisition initiated by nbStartScan.
nbSetAllLasers	Set all lasers ON/OFF for the entire system. **Not supported by some sensor models.**
nbGetCurtain	Returns real-time light curtain data from an individual sensor.
nbGetRangesAndCurtain	Returns real-time curtain and range data from an single sensor.
nbGetAllCurtain	Returns real-time curtain data from all installed sensors.
nbSetCurtainStatus	Called to mask out individual light curtain bits.
nbGetCurtainStatus	Returns information on whether a light curtain bit is enabled.
nbSetCurtainOffsets	Specifies encoder offsets for the light curtain data in the entire system.
nbGetCurtainOffsets	Returns the encoder offset table for the light curtain data in the system.

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

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

Input

OPENSCANNERST *opp	Pointer to a structure of type OPENSCANNERST. This structure contains two parameters required to initialize the NBLIB for board scanning: ***opp.IPAddress*** ASCII string, specifying the host IP address, such as "192.168.0.10". If this field is initialized as NULL, **nblib.dll** will detect the host. Specifying the host address is only necessary for the unlikely cases when there are more then a single host available at the same time. ***opp.hEvent*** 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 a manual resetable event by a call such as: `opp.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);` If this field is set to NULL, the library will not issue any asynchronous event notifications.
SCANNERPARAMS *user_sp	pointer to the structure containing user parameters for the scanner operation.

Return Value

BOOL bReturn	TRUE on success. FALSE on error. Call nbGetLastError 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 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 nbGetSystemDataInfo();

This function returns information on the data capabilities of the system.

Input

BXHEADDATAINFO *buffer	Pre-allocated output buffer. The size should be equal or greater than maxHeads * sizeof(BXHEADDATAINFO)
int maxHeads	Maximum number of BXHEADDATAINFO elements the supplied buffer can store.

Return

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

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 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 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 the laser in question.
int laser	The index of the laser whose status we want to modify.
BOOL enable	Desired status for the specified laser. When TRUE, the laser will be used in acquiring board scan data. When FALSE, the laser spot reading will always be Out of Range.

Return

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

Remarks

When the library is initialized, all lasers for all sensors are 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 from a given sensor. This value is defined by the sensor's model number.

Input

int sensor	The logical/system sensor index of the desired sensor.
int *numlasers	Pointer to an integer to store the number of valid lasers in the sensor. Depending on the sensor attached, thisvalue can be anywhere between 0 (all the lasers are disabled), and 8 for B8 sensors, or 16 for B16 sensors, or 23 for M24B sensor.

Return

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

Remarks

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 *encoderCount	Pointer to the destination variable.

Return

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

Remarks

When the MB Station 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 encoderCount	User value to replace the current encoder count.

Return

BOOL bReturn	TRUE on success; 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. **Current MB Station implementation does not support setting encoder value to anything other than zero.**

See also nbGetEncoderInfo

BOOL nbGetPhotocells(PHOTOCELLST *photocellsinfo);

This function fills a user buffer with the current photocell information and current encoder count. When the MB Station 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 *photocellsinfo	user buffer to be filled with the current photocells value and encoder count.

Return

BOOL bResult	TRUE on success FALSE on error. Call nbGetLastError 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 MB Station buffer can hold at most 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 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 a given sensor as read by the MB Station.

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 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 (isAvailble[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 the user during system calibration. Each entry in a laser range data buffer corresponds to an encoder pulse.

Input

SCANNEDOBJECTST *buffer	Pointer to a user allocated structure containing the scan buffer of the 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 photocell 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 SCANNOBJECTST fields are filled as follows:

buffer.firstscanline	holds index of the buffer line with the first board scan information.
buffer.lastscanline	holds the index of the buffer line with the last board scan information.
buffer.firstphotoline	holds index of the buffer line with the first valid photocells information.
buffer.lastphotoline	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, photocell values 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.

BOOL nbGetScannedObject2(SCANNEDOBJECTST *object);

This function provides the same benefits as nbGetScannedObject with the addition of light curtain support.

Input

SCANNEDOBJECT2ST *buffer	Pointer to a user allocated structure containing the scan buffer of the 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

For the remarks regarding the common functionality between nbGetScannedObject and nbGetScannedObject2 please see the description of the nbGetScannedObject function. The light curtain data array for each scan line of each sensor can be accessed in the following way:

```
unsigned short * curtain = (unsigned short *)buffer->scanline[ scanIndex ].lightcurtain;

for (i = 0; i < numHeads; ++i)
{
        if (isSensorPresent[i])    //was this sensor present in the system?
        {
                unsigned short * sensorCurtain = & curtain [i* NB_LIGHTCURTAIN_SIZE];
                for (j = 0; j < NB_LIGHTCURTAIN_CELLS; ++j)
                {
                        bitValue = sensorCurtain[bit / BITS_IN_WORD] & (0x8000 >> (bit
                        % BITS_IN_WORD))
                }
        }
}
```

For sensors which do not support light curtain output the data will be 0. This function does not have any performance drawbacks to nbGetScannedObject as the light curtain data is only transmitted from the MB Station/NPH-66 if the sensor supports it.

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

ASYNCEVENTST *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, the user can retrieve relevant information using this function. The 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 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().

Asynchronuous 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 B-series

Return

BOOL	TRUE on success FALSE on error. Call nbGetLastError 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. The user can provide his/her own offsets table that has been calculated to compensate for sensor 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.

Return

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

See Also nbSetOffsets

BOOL nbGetSensorMap(int *buffer,int buffersize);

This function is called to get a copy of the currently installed sensors map.

Input

int *buffer	Pointer to the user buffer to receive the copy of the sensor table.
int buffersize	Size of the destination buffer in bytes. Must be at least: `sizeof(int)*NB_MAXHEADS`

Return

BOOL bResult	TRUE on success FALSE on error.

Call nbGetLastError API function to retrieve more information on the nature of failure.

Remarks
The function is a complement of the function nbSetSensorMap. See nbSetSensorMap for more details on the sensor table.
See Also nbSetSensorMap

BOOL nbSetSensorMap(int *table,int tablesize);

This function is called to re-map the sensor indices from connector based (physical) to logical numbers based on the actual sensors hardware implementation.

Input

int *table	Pointer to the sensor table. The mapping is performed as logical = table[physical]. The table must contain unique values for all sensors.
int tablesize	Size of sensor table in bytes. Must be at least: `sizeof(int) * NB_MAXHEADS`

Return

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

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 *nblib.dll* library returns unsuccessfully.

Input

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

Return

int lasterror 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 MB Station memory.

Input

SCANNERPARAMS *user_sp — Pointer to the MB Station concentrator parameters structure.

Return

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

Remarks
Upon boot up the MB Station 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 **MB Station** 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 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. The MB Station logs current encoder count on each photocell transition, light to dark and vice versa. Upon initialization of the scanner, the user specifies the photocell index to be used for this data logging. Once the transition occurs, the MB Station will transmit a PCSTATEST structure to the client computer over the Ethernet connection using UDP protocol. Upon receiving this UDP packet the nblib.dll will issue the ASYNC_EVENT_LOG_PC_INFO event. Subsequently, the 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 form the MB Station 1 - issue the *TCP/IP request to poll for the data from the MB Station

Return

BOOL bResult	TRUE on success FALSE on error. Call nbGetLastError 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 when the chain is not moving at that time or a confirmation is needed.

BOOL nbStartScan(void);

Sends a TCP/IP command to the MB Station unit to trigger a collection of scan frames upon each encoder pulse, up to *maxwidth* scan frames. This command is recognized by the MB Station unit only in the Software Trigger Mode (see Modes of Operation). The 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 to retrieve more information on the nature of failure.

Remarks
See also nbStopScan()

BOOL nbStopScan();

This call is used in Software Triggered Mode only (see Modes of Operation). Upon issuing this TCP command, the MB Station will terminate data acquisition of the current segment, and subsequently issues a Board_Ready event regardless of the 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 are not available to the user.

Input

void

Return

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

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

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

This function is called to initialize a serial port on the MB Station to enable serial communication between the MB Station and the 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 to retrieve more information on the nature of failure.

Remarks

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

void nbResetMB Station(void);

Soft reset the MB Station unit.

Input

void

Return

void

Remarks

Current MB Station implementation does not support this feature.

BOOL nbGetCurtain(int head, unsigned short * curtain);

Returns real-time light curtain data from an individual sensor.

Input

int head	Logical head number
unsigned short * curtain	User buffer to be filled with the curtain data. Must be sized NB_LIGHTCURTAIN_SIZE * sizeof (unsigned short).

Return

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

Remarks

The curtain buffer is fixed and must be at least the size sizeof(short)*NB_LIGHTCURTAIN_SIZE.

A possible calling sequence would be:

```
curtain = (unsigned short*) malloc(sizeof(short)*NB_LIGHTCURTAIN_SIZE);

if (nbGetCurtain(head,curtain))
{
        //  Process data
}
else
{
        //  Process error
}
```

BOOL nbGetRangesAndCurtain(int head, unsigned short * ranges, unsigned short * curtain);

Returns real-time light curtain and range data from an individual sensor.

Input

int head	Logical head number
unsigned short * ranges	User buffer to be filled with the ranges. Must contain space for NBMAXSPOTS ranges.
unsigned short * curtain	User buffer to be filled with the curtain data. Must be sized NB_LIGHTCURTAIN_SIZE * sizeof (unsigned short).

Return

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

Remarks

The curtain buffer is fixed and must be at least the size sizeof(short)*NB_LIGHTCURTAIN_SIZE. The range buffer must contain space for NBMAXSPOTS ranges. A possible calling sequence would be:

```
curtain = (unsigned short*) malloc(sizeof(short)*NB_LIGHTCURTAIN_SIZE);
ranges = (unsigned short*) malloc(sizeof(short)*NB_MAXSPOTS);

if (nbGetRangesAndCurtain(head,ranges, curtain))
{
        //  Process data
}
else
{
        //  Process error
}
```

BOOL nbGetAllCurtain(int numHeads ,int * isAvailable, unsigned short * curtain);

Returns real-time light curtain data from all installed sensors

Input

int numHeads	Number of sensors in the system (0...NB_MAXHEADS)
unsigned int * isAvailable	Reports whether a sensor is present
unsigned short * curtain	Reports light curtain data

Return

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

Remarks

The isAvailable buffer must contain at least numHeads elements. The curtain buffer must contain at least numHeads*NB_LIGHTCURTAIN_SIZE elements.

```
curtainBits = NB_LIGHTCURTAIN_SIZE*BITS_IN_WORD;
isAvailable = (int*)malloc(sizeof(int)*numHeads);
curtain = (unsigned short*)malloc(sizeof(unsigned short)*numHeads*NB_LIGHTCURTAIN_SIZE);

if (nbGetAllCurtain(numHeads,isAvailable, curtain))
{
        for (i = 0; i < numHeads; ++i)
        {
```

```
                if (isAvailble[i])    //was this sensor present in the system?
                {
                        sensorCurtain = &curtain[i* NB_LIGHTCURTAIN_SIZE];
                        for (j = 0; j < curtainBits; ++j)
                        {
                                bitValue = sensorCurtain[bit / BITS_IN_WORD] & (0x8000 >>
                                        (bit % BITS_IN_WORD))
                        }
                }
        }
}
else
{
        //process error
}
```

Note that the range data for a missing sensor will be set to LMI_OUTRANGE, while the light curtain data will be set to 0.

BOOL nbSetCurtainStatus (int head, unsigned short * curtainMask)

This function specifies a mask, which will be ANDed with the light curtain data returned by the specified sensor.

Input

int head	Index of the head, for which the curtain status is to be changed
unsigned short * curtainMask	Light curtain mask buffer

Return

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

Remarks

'curtainMask' must point to a buffer NB_LIGHTCURTAIN_SIZE words in size. When the library is initialized, all of the curtain bits are enabled. The bit order is expected to be exactly the same as the one in the curtain buffer returned by the sensor.

BOOL nbGetCurtainStatus (int head, unsigned short * curtainMask)

This function specifies a mask, which will be ANDed with the light curtain data returned by the specified sensor.

Input

int head	Index of the head, for which the curtain status is to be returned
unsigned short * curtainMask	Light curtain mask buffer

Return

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

Remarks

'curtainMask' must point to a buffer NB_LIGHTCURTAIN_SIZE words in size.

BOOL nbSetCurtainOffsets (int* curtainOffsets, int bytesize)

This function is called to specify the light curtain offsets table used by nbGetScannedObject2. The table must have one entry for each light curtain cell(bit) for all sensors.

Input

int * curtainOffsets	Pointer to the offsets table
Int bytesize	Size of the offsets table in bytes. Should be sizeof(int)*NB_MAXHEADS*NB_LIGHTCURTAIN_CELLS 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 to retrieve more information on the nature of failure.

Remarks

When the library is initialized, the offsets are all set to zero. Note that the offsets for the unused light curtain bits in the scanned object buffer should not be specified. For each sensor the number of offset values should be exactly NB_LIGHTCURTAIN_CELLS

BOOL nbGetCurtainOffsets (int* curtainOffsets, int bytesize)

This function is called to retrieve the light curtain offsets table used by nbGetScannedObject2.

Input

int * curtainOffsets	Pointer to the user buffer receiving the offsets table
Int bytesize	Size of the user buffer in bytes. The buffer should be at least sizeof(int)*NB_MAXHEADS*NB_LIGHTCURTAIN_CELLS

Return

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

5.2 NbLib API Errors

Table 5.2 – NBLib API Errors

NB_NOERROR	No error has occured	0
NB_ERROR_SOCKET		1
NB_ERROR_NOHOST	No host is detected. Ensure that your MB Station 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 MB Station 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 MB Station	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

6 MB Station and Sensor Parameter Structure

Using LMI's API interface the user software is responsible for setting up the scanner parameters upon establishing a network connection with the MB Station 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. The rest of them are not employed and have no impact on scanner functionality. The following table describes the use of these parameters:

Table 6 – Scanner Parameter and Description

SCANNERPARAMS	Description
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 the 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.
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.
trail_holdscan	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.
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..5).	
escale	Encoder scaling factor, i.e. value of 4 means that the data will be stored at the resolution of 4 encoder pulses (Every 4th pulse will be stored).	
pcindex_len	Specifies photocell input index (0..5) used for length measurement (must be different form the *triggerindex*).	
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..5) used for manually stopping scan (must be different form the *triggerindex & lengthindex or it will be disabled*). This is only for use in mode 1.	
timer_freq	Specifies frequency at which the MB Station will poll the sensors for data. 0 = 1000Hz 1 = 1500Hz 2 = 2000Hz The default value is 1000Hz.	
encoder_mode	ENCODER_MODE_ONE_COUNT = 0	Encoder is sampled at the rising edge of channel A. This produces one count per set of quadrature signals.
	ENCODER_MODE_TWO_COUNTS = 1	Encoder is sampled at the rising and the falling edges of channel A. This produces two counts per set of quadrature signals.
	ENCODER_MODE_FOUR_COUNTS = 2	Encoder is sampled at the rising and falling edges of channels A and B. This produces four counts per set of quadrature signals.
	Note: Increasing the encoder frequency does not change the data sampling frequency. If the effective encoder frequency exceeds the sampling frequency ('timer_frequency' field in SCANNERPARAMS) encoder counts will be skipped.	

7 System Setup

7.1 OEM 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, the 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 uses 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.

7.2 MB Station IP Address Setup

WARNING: The client's subnet address must match the one used by the MB Station in order to connect to it. There are no tools provided for discovering MB Station subnet address if it is lost.

MB Station is setup with a default IP address **192.168.0.151** and subnet mask **255.255.255.0**. Your networks IP address may be quite different from the default and therefore the MB Station IP and subnet mask may need to be permanently modified to become a part of your network. The procedure requires a valid Ethernet connection to the MB Station.

Tools

Hardware

Desktop or a Notebook PC with Ethernet 10/100 Mb adapter installed
MB Station

Software

Windows XP SP2 Operating system
B-Series OEM CD(kIPConfig program)

Procedure

1. Modify your network settings to match the default subnet of the MB Station. (for example IP: 192.168.0.100, mask: 255.255.255.0)
2. Make sure that the B-Series OEM CD is installed in a local folder.
3. Open the kIPConfig program and select the network interface that is used to connect to the MB Station.
4. Click 'Enumerate' and wait for serial numbers to show up on the left side of the window.

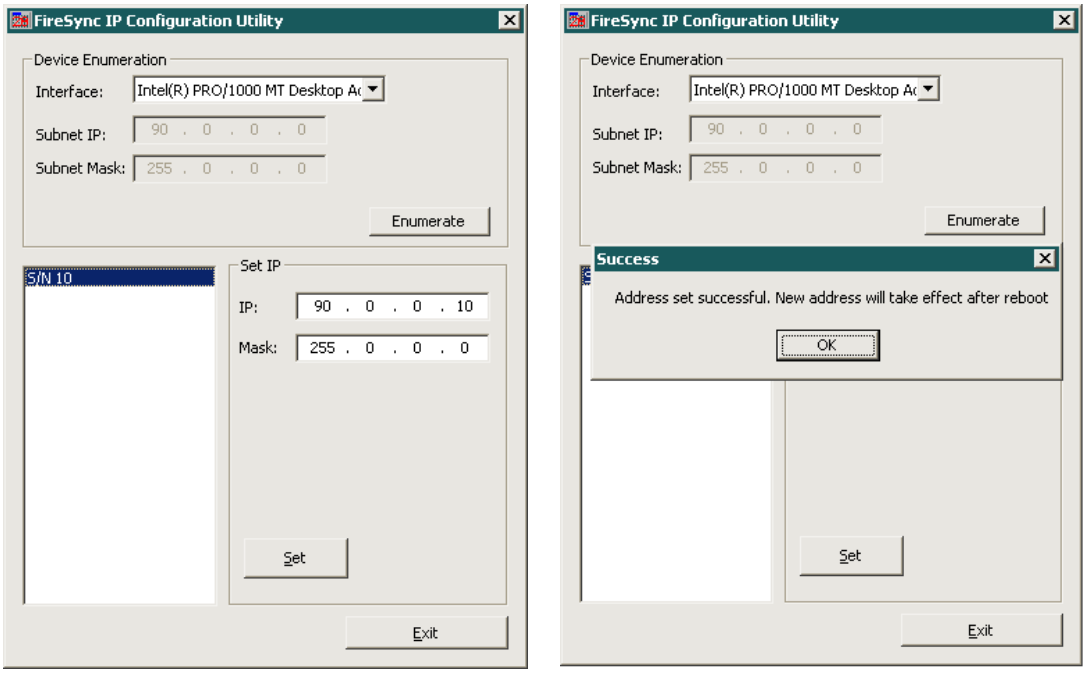


Figure 7.1 – FireSync IP Configuration Utility Software

5. Enter the desired IP and subnet mask in the ‘IP’ and ‘Mask’ fields.
6. Click the ‘Set’ button and wait for a change to occur. Do not power down the MB Station during the update process. When a Success window comes up, click ok and re-power the station.
7. Connect to the MB Station using the new network settings.

7.3 MB Station Firmware Update

MB Station firmware upload is done over the *Ethernet connection*. The following procedure assumes that the network settings are configured correctly. Please refer to section 8.3 for more information on the IP/Subnet mask setup.

Tools

Hardware

Desktop or a Notebook PC with Ethernet 10/100 Mb adapter installed
MB Station

Software

Windows XP SP2 Operating system
B-Series OEM CD(kUpdate program)

Procedure

1. Power on the MB Station. Allow 30 seconds for the station to boot.
2. Run the **kUpdate.exe** program and verify that the MB Station is listed in the 'Devices' tab.

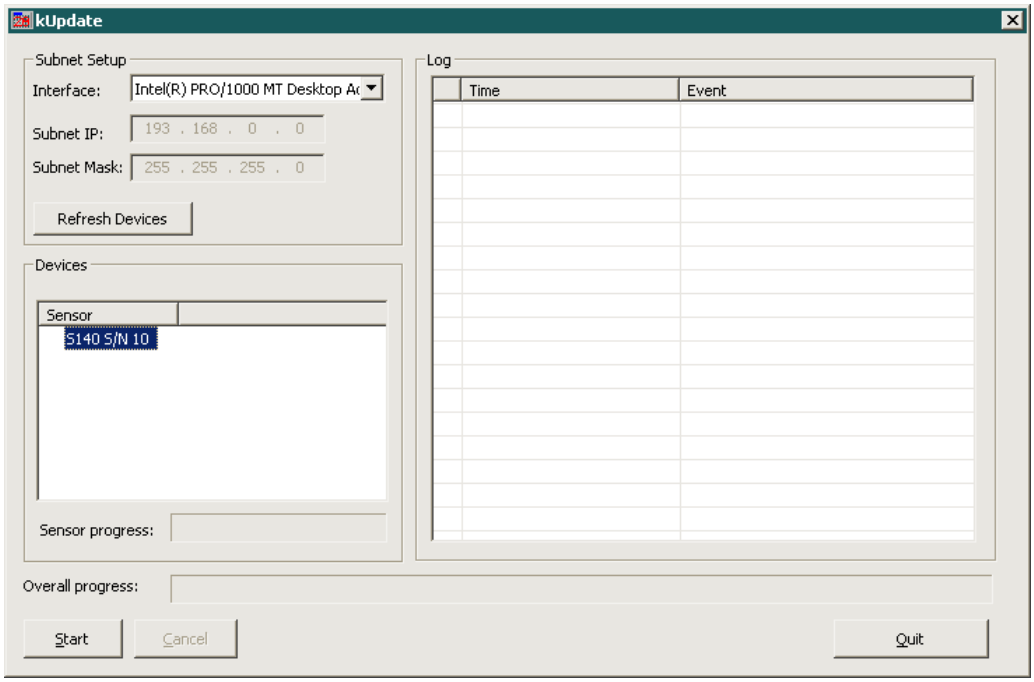


Figure 7.2 – kUpdate Program, used for updating firmware

3. Click the 'Start' button. A window will pop up.
4. Choose the FPGA firmware to upload. Select the 'Open' button.
5. A new window will pop up requesting for the DSP firmware to upload.
6. Select the DSP from the folder and select 'Open'.

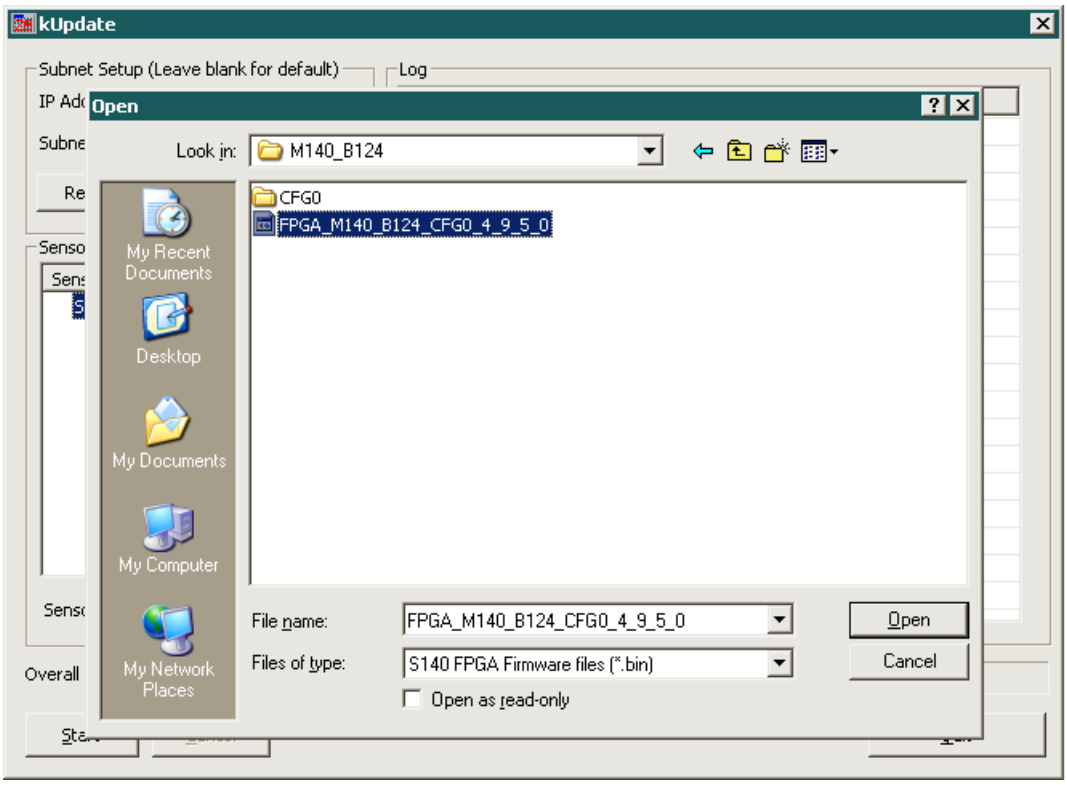


Figure 7.3 – kUpdate, program used to update Firmware and FPGA

7. kUpdate will start uploading the FPGA and DSP. See below.

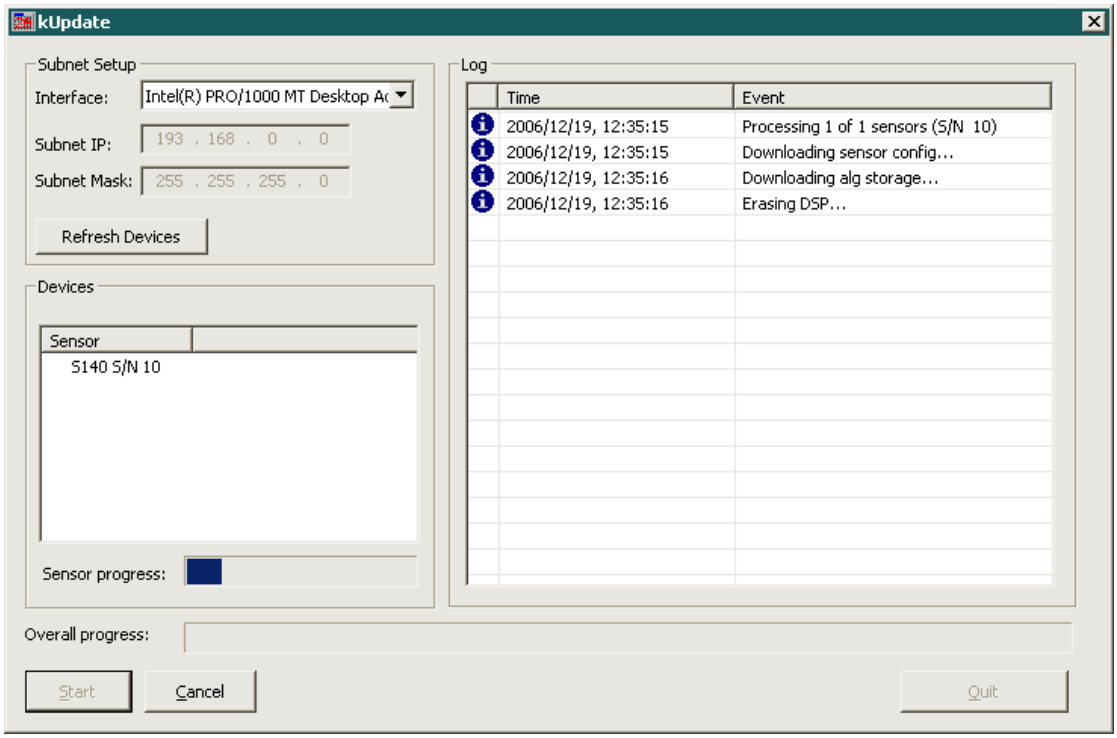


Figure 7.4 – kUpdate program, updating the FPGA and DSP

8. Wait for the files to be uploaded and the MB Station to be rebooted.
9. Close kUpdate and re-power the MB Station. This step is required due to additional diagnostics performed by kUpdate after the initial reboot is complete.

7.4 Sensor Firmware Upload Procedure

Tools

Hardware

Desktop or a Notebook PC with Ethernet 10/100 Mb adapter installed.
MB Station

Software

Windows '98/NT/2000/XP Operating system
BxLoad.exe
Nblib.dll

Procedure

1. Connect all the sensors to the MB Station
2. Connect MB Station to your network using a standard CAT5 Ethernet cable or directly to a computer using a crossover CAT5 Ethernet cable.
3. Power up the MB Station. The unit will boot up in about 30 seconds and is ready to operate.
4. Run the BxLoad.exe utility on your PC.
5. Click the 'Connect' button within the dialogue box.

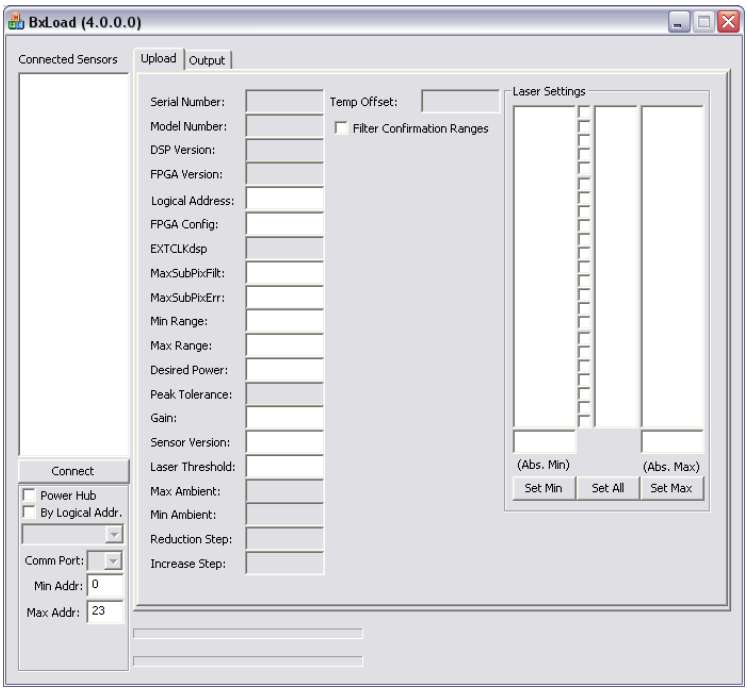


Figure 7.5 – BxLoad, Initial screen

6. The program will detect all operational sensors connected to the MB STATION unit. Once finished, the list of detected sensor serial numbers will appear in the left pane of the dialogue window.

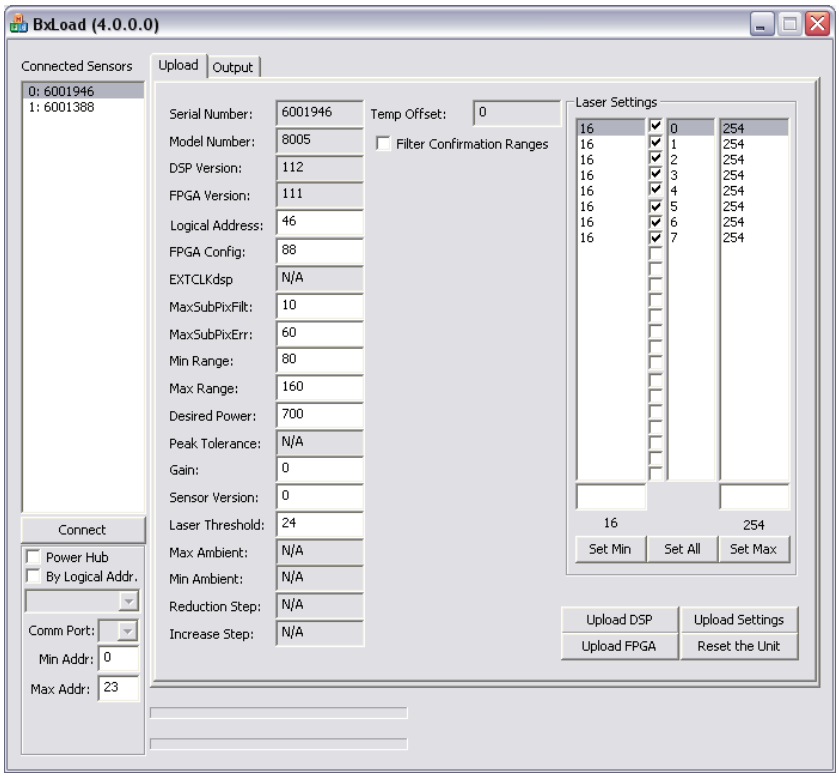


Figure 7.6 – BxLoad, Connected

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. The 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.

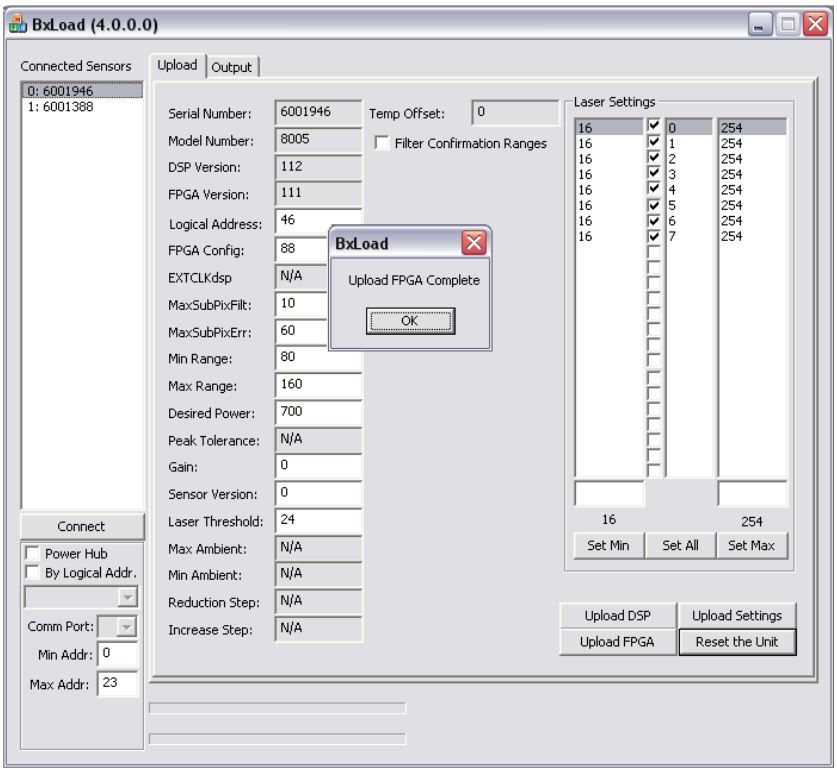


Figure 7.7 – BxLoad, Upload complete

9. Press ‘OK’. The program will return to the initial screen.
10. 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.
11. Next BxLoad will display a DSP Parameters dialogue box. These parameters are modified either during initial setup or by a qualified LMI field service technician.
12. To accept the settings and upload them, click the ‘Upload Settings’ button. BxLoad will then automatically start uploading code to the sensor. Progress is indicated by a progress bar. Once finished a confirming dialogue message box appears. Press OK. IF THE UPLOAD FAILS DO NOT RESET OR POWER DOWN THE SENSOR, please try the upload again.
13. At this point the selected sensor is uploaded with both new FPGA and DSP code.
14. Click the ‘Reset the Unit’ button to reset the sensor so the settings can take effect. You will now return to the initial screen and you can select another sensor to upload to.
15. Repeat the procedure, steps 7 through 12 for all the remaining sensors.
16. 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.
17. 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 1 through 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 to a sensor: Enumeration and Override Enumeration. Override Enumeration method is used to detect a sensor specifying the last three digits of the serial number. This 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 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.

7.5 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:

nblib.dll	LMI API library
factor.dat	A text file containing desired scanner parameters. This file can be edited/modified using any text editor to test different scanner parameters. If this file is **not present** the NbTest.exe will create one using default parameters once it is opened. (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. The user can run a straight edge board under the scanner (see Scans Tab) and once the board data is available, pressing 'Recalculate' 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 the '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.

7.5.1 System Tab

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

Connect

Pressing this will establish Ethernet communication with the MB Station host concentrator. The Nbtest application also receives the system status indicating the 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 to the right of the Connect button. When connected this window will be green and say “No Error”.

Modify Logical Sensors Map

MB Station is using a user defined logical table of connector vs. system sensor map allows the user to connect sensors to the MB Station arbitrarily. This table is permanently stored in the MB Station. NBTEST allows the user to modify this table and permanently upload new tables to the MB Station (see nbSetSensorMap and nbGetSensorMap API functions). Use the scroll controls to set sensor index vs. connector index and then press "Remap" to permanently upload the map to the MB Station.

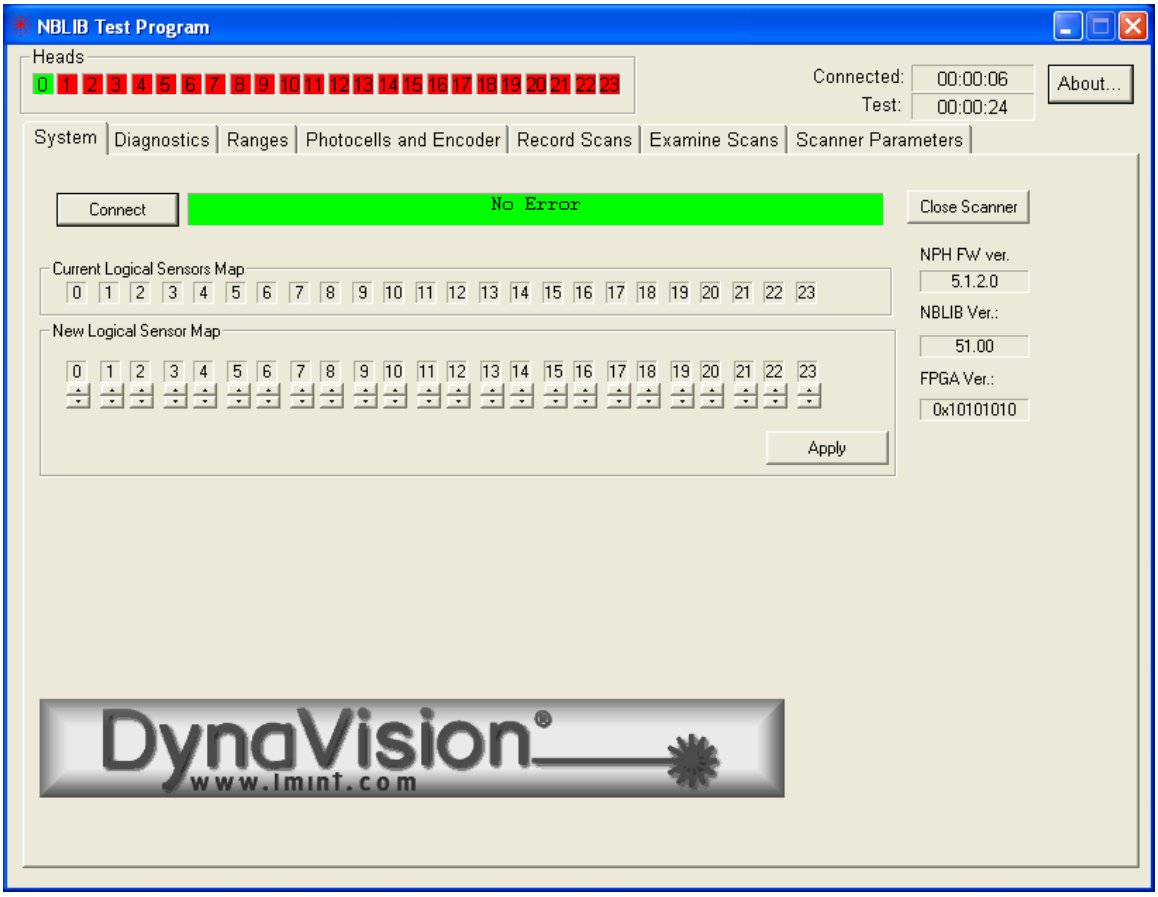


Figure 7.8 – NbTest, System Tab

7.5.2 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 a 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.

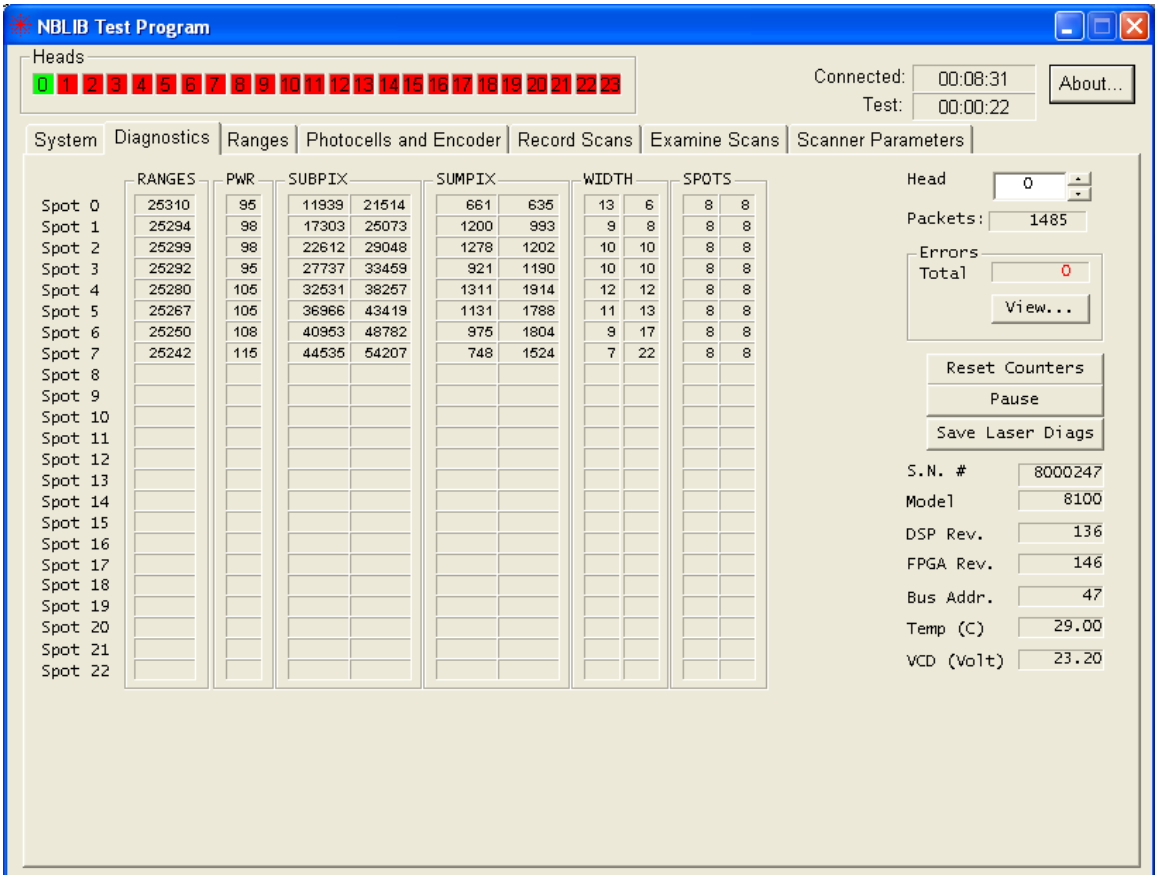


Figure 7.9 – NbTest, Diagnostics Tab

7.5.3 Ranges

The Ranges Tab displays both numerical and a bar graph representation of current range readings. API call nbGetRanges(...) is used.

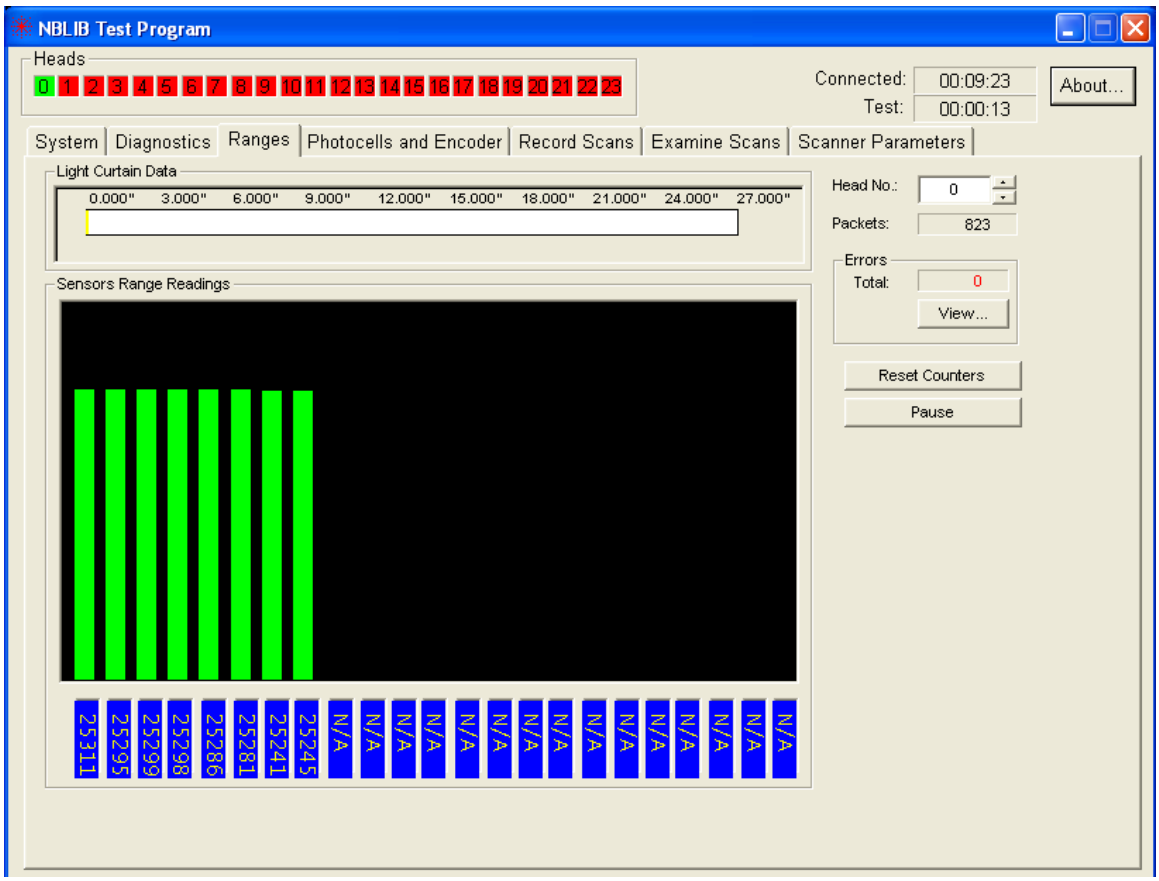


Figure 7.10 – NbTest, Diagnostics Tab

7.5.4 Photocells and Encoder Tab

This dialog window displays real-time encoder and photocell statuses. Current MB Stations supports up to 6 photocell inputs. 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 in a text format.

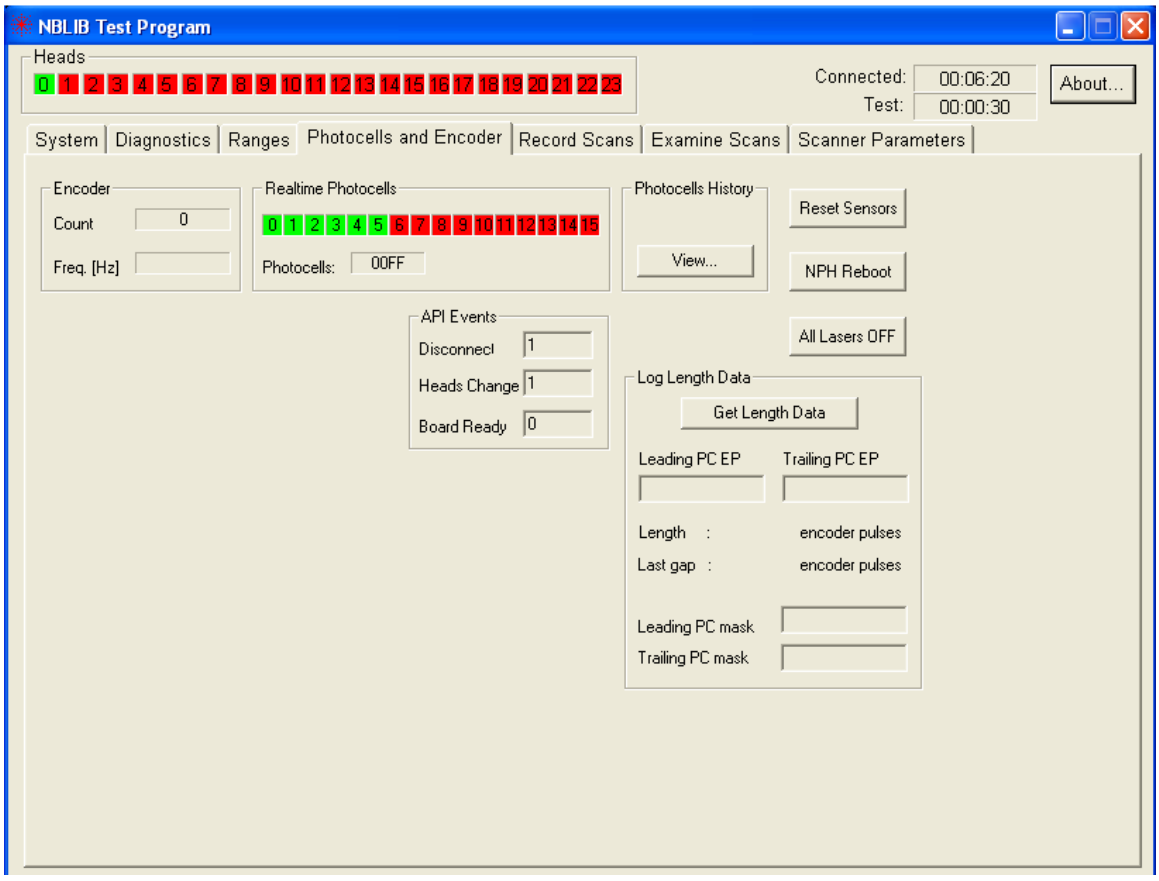


Figure 7.11 – NbTest, Photocells and Encoder Tab

7.5.5 Record Scans Tab

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

The right pane displays related photocell statuses. Green represents the open state of a photocell and yellow the blocked state.

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

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

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

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

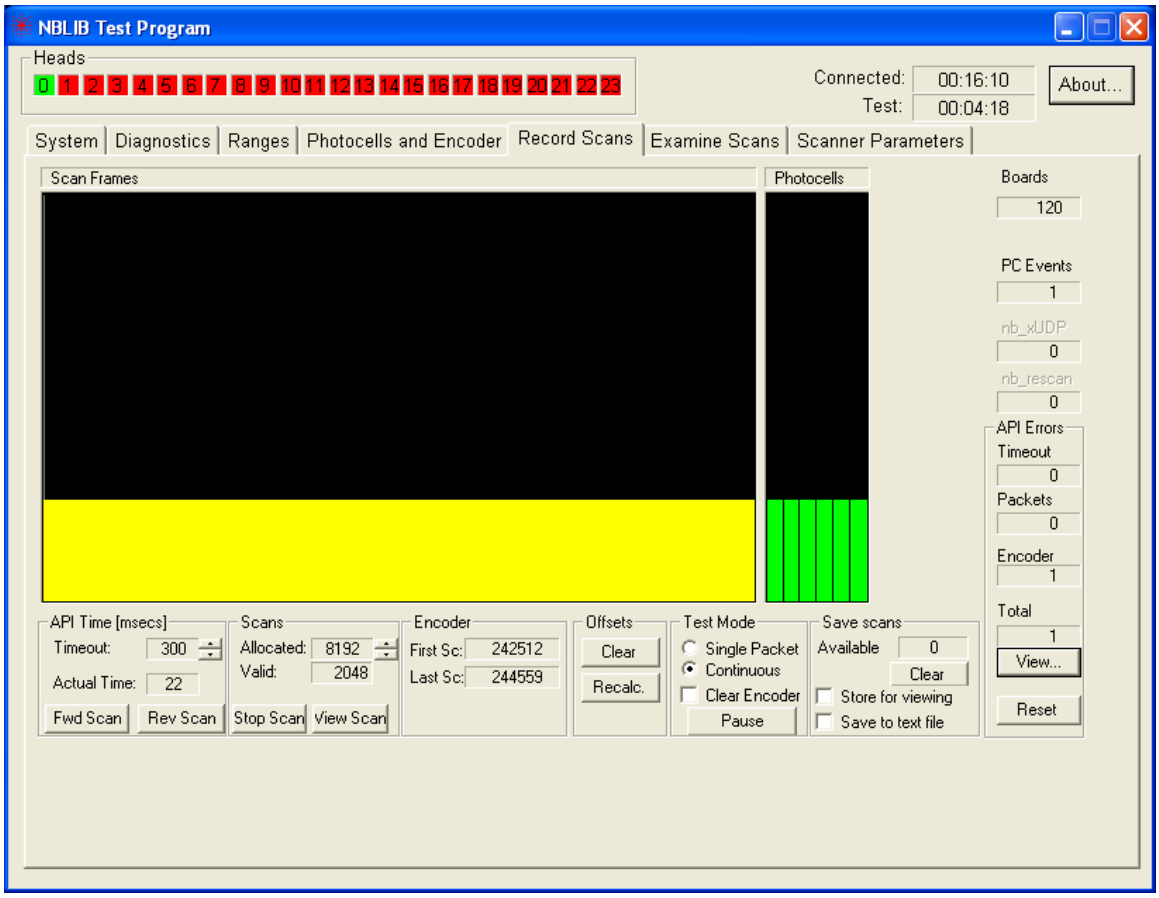


Figure 7.12 – NbTest, Scans Tab

7.5.6 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.

-">" ">>" and "<<""<" 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.

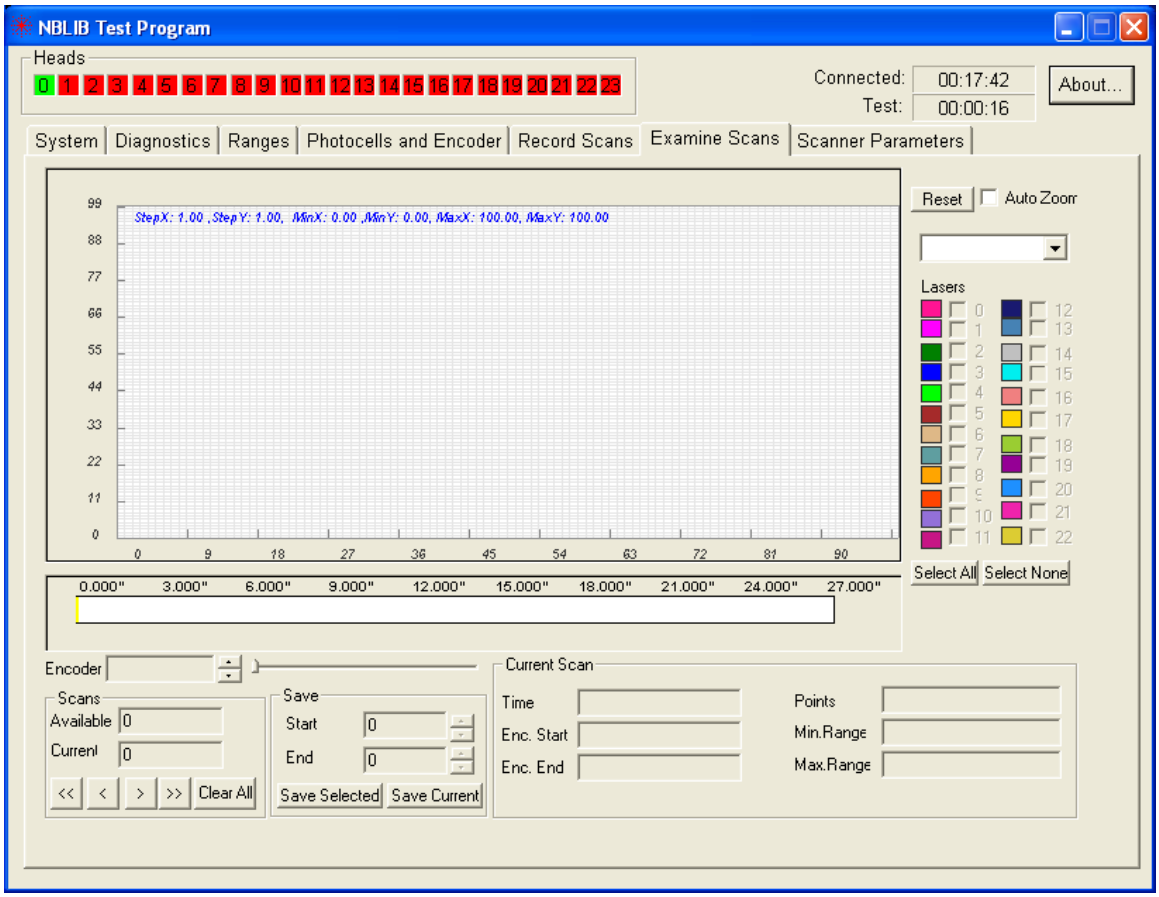


Figure 7.13 – NbTest, Scans Tab

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

7.6 BxSystemConfig Utility

BxSystemConfig utility is a Windows '9x/NT/2000/XP which can be used to configure B-Series sensors, once they are placed into a system. The functionality of this application is a subset of BxLoad, which is also distributed. However, because many expert level settings are not present, BxSystemConfig is the preferred application for the end user. BxSystemConfig allows users to configure the serial bus address and specify whether the sensor is in the top or bottom row. Note that top/bottom configuration is currently only required for B900 sensors.

7.6.1 Uploading settings

Tools

Hardware

Desktop or a Notebook PC with Ethernet 10/100 Mb adapter installed.
MB Station

Software

Windows '98/NT/2000/XP Operating system
BxSystemConfig.exe
Nblib.dll

Procedure

1. Connect all the sensors to the MB Station
2. Connect MB Station to your network using a standard CAT5 Ethernet cable or directly to a computer using a crossover
3. Power up the MB Station. The unit will boot up in about 30 seconds and is ready to operate.
4. Run BxSystemConfig application on PC.
5. Open File->Connect menu item.

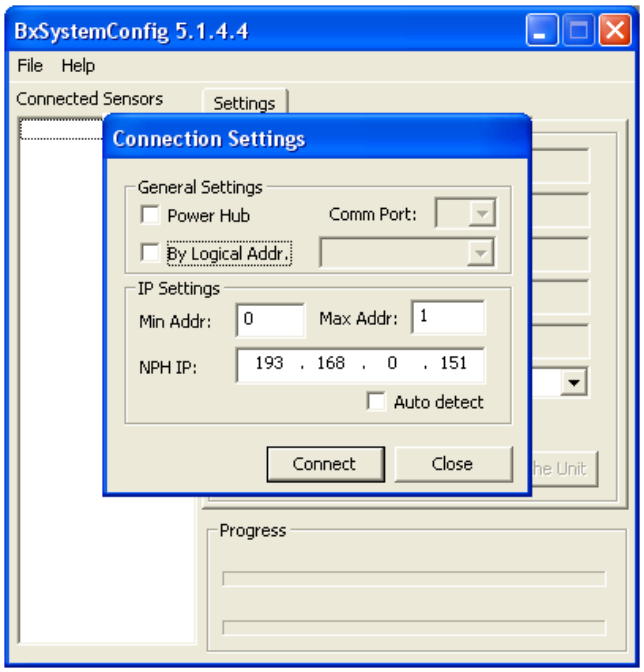


Figure 7.14 – Bx System Configuration Utility program

6. Input MB Station IP address. Select the sensors port range or logical address range and press ‘Connect’.

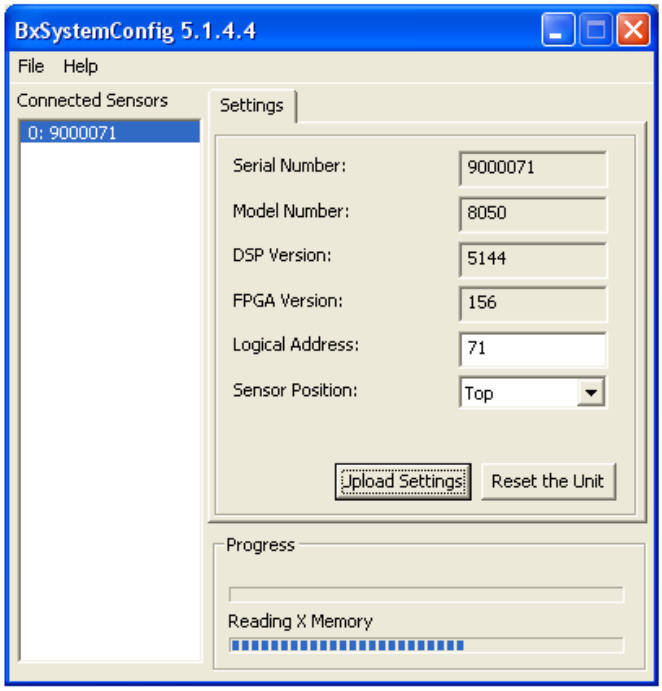


Figure 7.15– Bx System Configuration Utility program, Uploading settings

7. Select sensor serial number.
8. If required, modify the Logical Address such that it is unique across the entire system. The valid range is 1..254. By default the Logical Address is set to the last two digits of the serial number.
9. If required, modify the Sensors Position setting. For B900 sensors the value must accurately specify whether the sensor is in top or bottom row. The setting currently has no effect on other B-series sensor models.
10. Press Upload Settings.
11. Press Reset the Unit for settings to take effect.

8 Getting Started

This section outlines the steps required to get a B-Series system with an MB Station running. As this is only a brief overview, it may direct the user to other sections of this manual for further information.

8.1 Powering Up

The user is required to provide cabling to power the MB Station as well as the cables to connect and power the sensors. Please refer to **Section 4** for the instructions and diagrams. Standard CAT5 Ethernet cable can be used to connect the MB Station to the PC. The following steps can be taken to verify that the system has been powered on correctly.

1. Connect one or more B-Series sensors using a ‘Bx Power/Data Cable’ and power the MB Station as described in **Section 4.**
2. Verify that the green power LED comes on. If the photocells are not connected, all six photocell LEDs should turn green as well.
3. Verify that the connected sensors have been powered correctly:

    - If using sensors with LCD displays (B8,B8A, M24B), please verify that the LCD comes on with correct information.
    - If using sensors without LCD displays (Bx00), please confirm that the lasers come on in sequence. This confirms that they have been powered on and synchronized correctly by the station.

8.2 Using Diagnostic Software

MB Station is setup with a default IP address **192.168.0.151** and subnet mask **255.255.255.0**. To get started, please configure your network card settings to match the subnet (for example IP: 192.168.0.100 , mask: 255.255.255.0). For the instructions on changing the MB Station IP/subnet mask please refer to **Section 8.3.** The following instructions assume that all steps in **Section 9.1** have been completed successfully.

1. Copy the contents of the B-Series OEM CD into a local folder.
2. Start the **nbTest** application (located in **API\bin**)
3. Open the 'Scanner Parameters' tab.

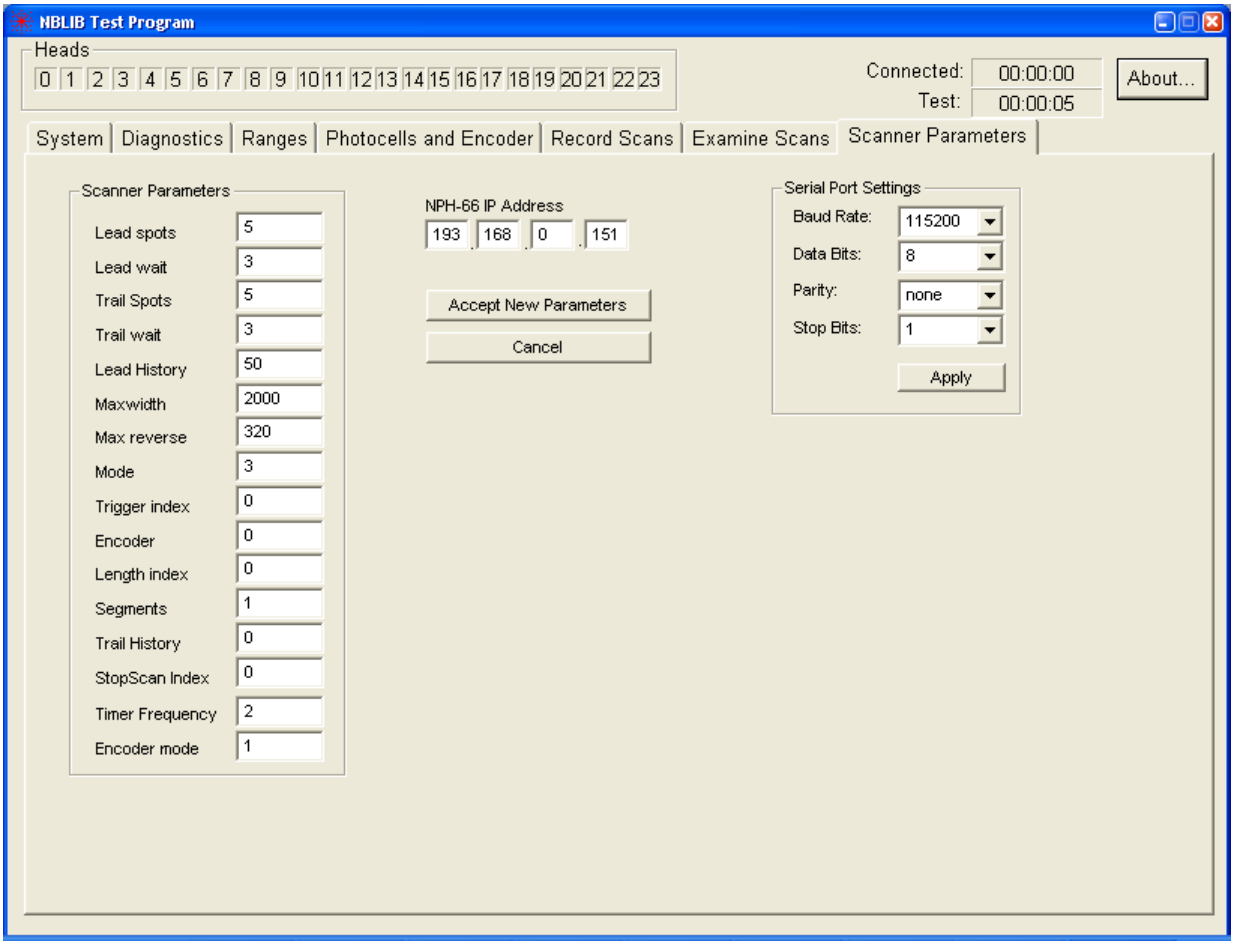


Figure 8.1 – NBLIB Test Program, Scanner Parameter settings.

4. Enter the MB Station IP address into the 'NPH-66 IP Address' field and press 'Accept New Parameters' button.
5. Open the 'System' tab and press the 'Connect' button.

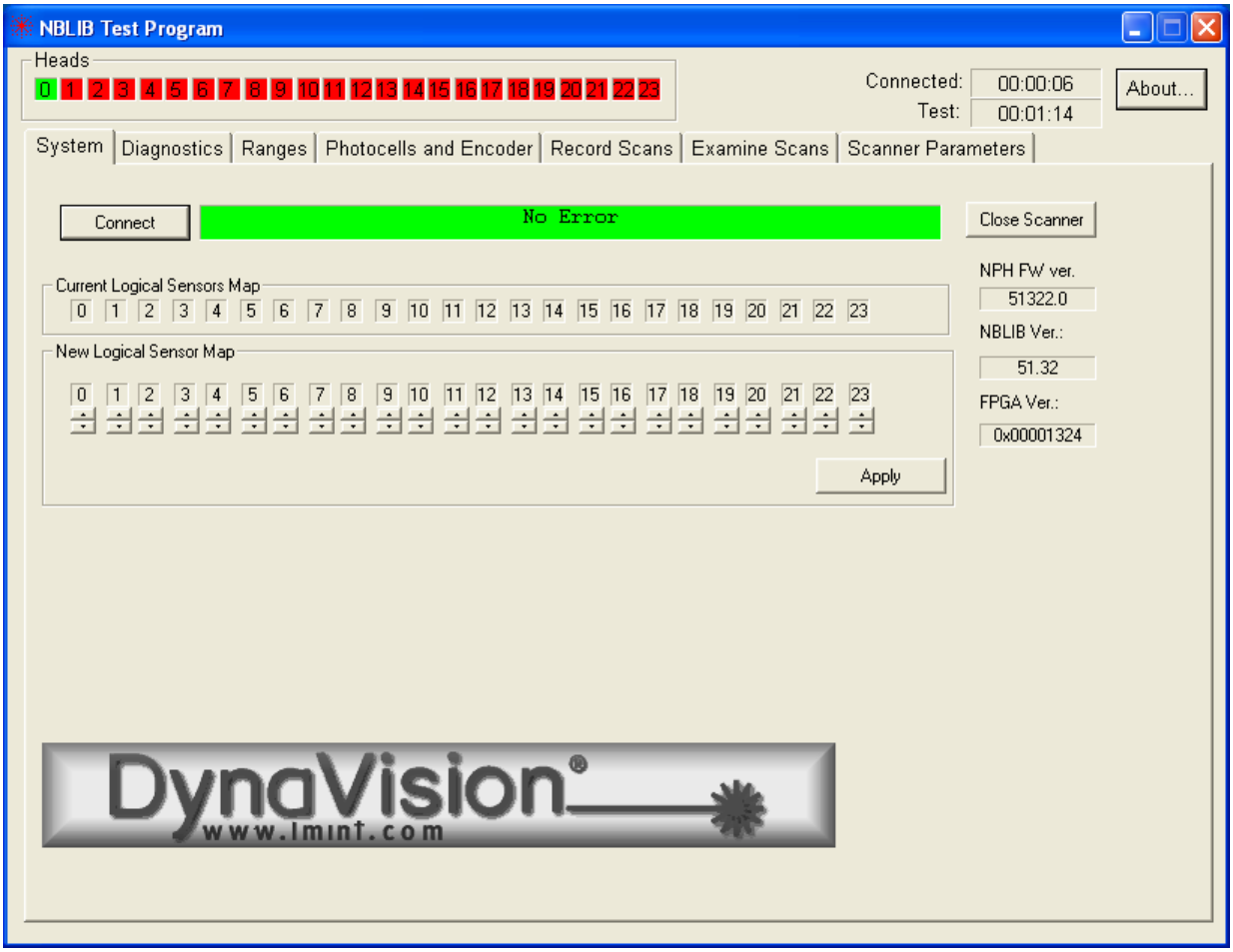


Figure 8.2 – NBLIB Test Program, System Tab view

6. Verify that the status bar shows ‘No Error’ and that the connected heads are marked green in the ‘Heads’ indicator bar.
7. Select the ‘Diagnostics’ tab and select the index of one of the connected heads.
8. The ‘Ranges’ column shows the distance from the face of the selected sensor to the target. If a given laser is not hitting a target or a target is out of range the reported value is **32768.**
9. The “S/N #” field should match the serial number label on the sensor housing.

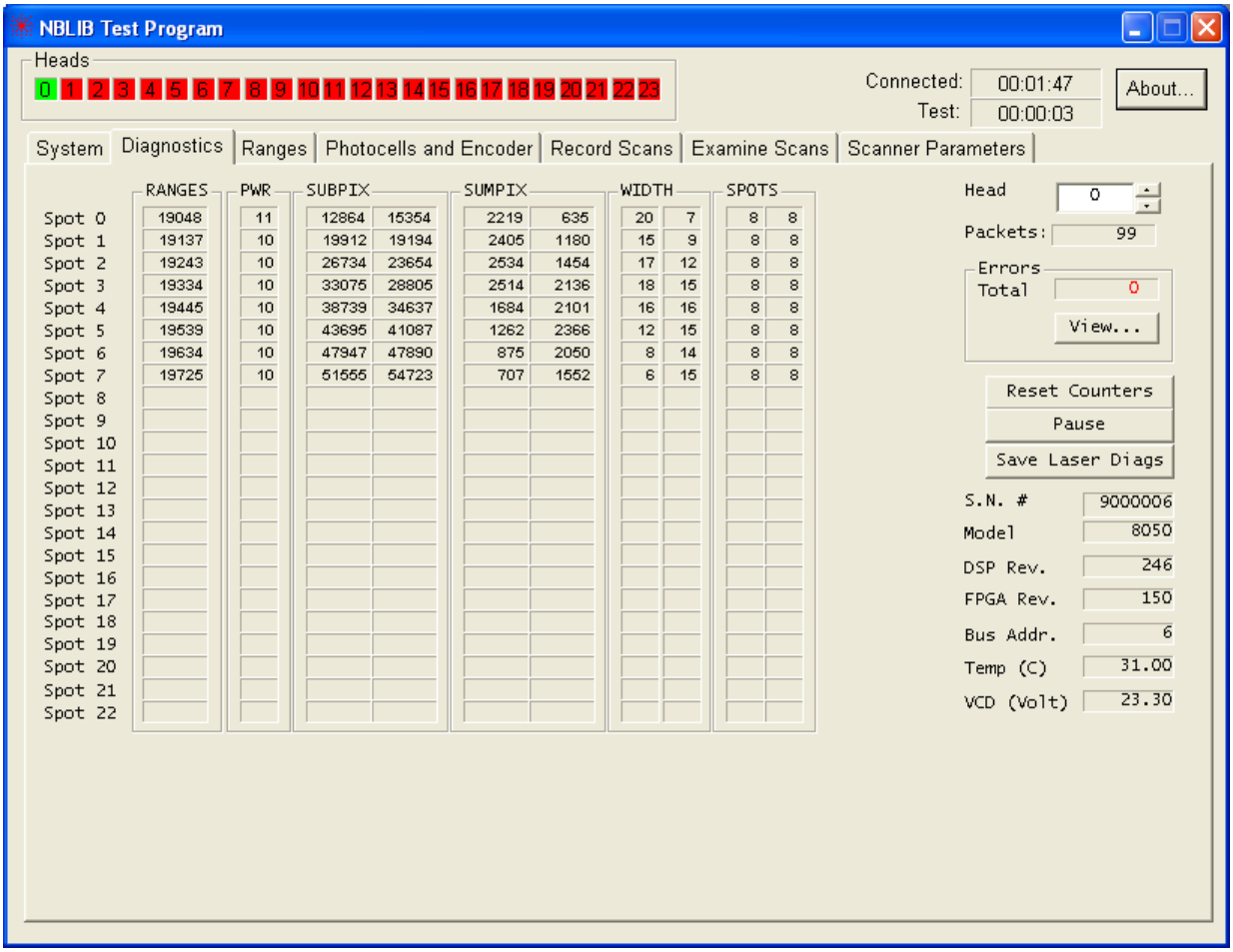


Figure 8.3 – NBLIB Test Program, Diagnostic Tab view

8.3 Using the API

The MB Station is fully compatible with software written for NPH-66. The B-Series OEM CD contains two sample projects to get the user started with software development. The sample projects can be built using Visual Studio 2003. The **Mt_test** project demonstrates the basic steps required to establish a connection with MB Station and start receiving sensor data. The **Curtain_test** project has the same structure as mt_test, however, it also provides sample code for parsing the light curtain data (for the sensors that support it).

9 Warranty

9.1 Warranty Policies

The sensor is warranted for two years from the date of purchase from LMI Technologies Inc.. Products that are found to be non-conforming during there warranty period are to be returned to LMI Technologies Inc. The shipper is responsible for covering all duties and freight for returning the sensor to LMI. It is at LMI's discretion to repair or replace sensors that are returned for warranty work. LMI Technologies Inc. warranty covers parts, labor and the return shipping charges. If the warranty stickers on the sensors are removed or appear to be tampered with, LMI will void the warranty of the sensor.

9.2 Return Policy

Before returning the product for repair (warranty or non-warranty) a return material authorization (RMA) number must be obtained from LMI. Please call LMI to obtain this RMA number. Carefully package the sensor in its original shipping materials (or equivalent) and ship the sensor prepaid to your designated LMI location. Please insure that the RMA number is clearly written on the outside of the package. With the sensors include the address you wish this shipment returned to, the name, email and telephone number of a technical contact should we need to discuss this repair and details of the nature of the malfunction. For non-warranty repairs, a purchase order for the repair charges must accompany the returning sensor. LMI Technologies Inc. is not responsible for damages to a sensor that is the result of improper packaging or damage during transit by the courier.

10 Getting Help

If you wish further help on the component or product contact your distributor or LMI directly. Visit our website at www.lmi3D.com for the agent nearest you.

For more information on Safety and Laser classifications, contact:

U.S. Food and Drug Administration
Center for Devices and Radiological Health
WO66-G609
10903 New Hampshire Avenue
Silver Spring, MD 20993-0002