CHAPTER 7
USING ADJUSTABLE FOCUS HEADS AND AUTO FOCUS FEATURES

Section I: Using the Adjustable Focus –AF Head

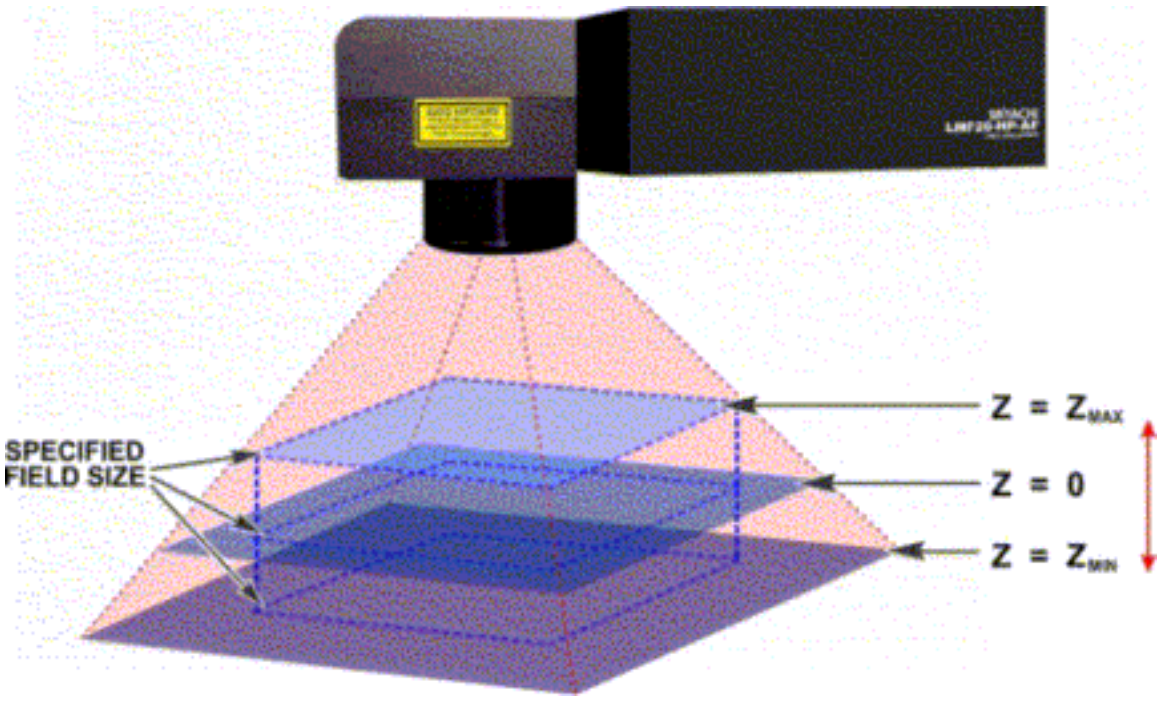
Note: The –AF Head and feature set requires firmware versions ≥ 6.x

Note: Real Time Focus Offset Adjustment will not take effect while a job is executing. Focus adjustments included in WinLase job objects will be executed in real time.

How the –AF (Adjustable Focus) Head Works

The purpose of the Adjustable Focus head (hereinafter called *–AF head*) is to allow the focused beam plane to shift to accommodate parts of various shapes and positions without physically moving the part or laser head.

The –AF head consists of a regular 2-axis scanner system with a focus shift module attached. The focus shift module moves a lens axially and causes the focus plane to shift. It is only possible to mark in flat planes that are located between the minimum and maximum shift position. Each lens combination has a maximum position (called Z_{MAX}) close to the lens and a minimum position (called Z_{MIN}) at the furthest position from the lens. The native focus point is designated $Z = 0$. For simplicity the field size is specified at one value across the entire range.



Using the –AF Head in WinLase

There are two types of focal adjustments made – individual *Object Based Adjustments*, and global *Autofocus Adjustments*. The sum of the two adjustments must be between $Z_{MIN} \leq Z \leq Z_{MAX}$. When the machine is first turned on the global adjustment defaults to $Z = 0$.

Focus shift adjustments can be made for several reasons.

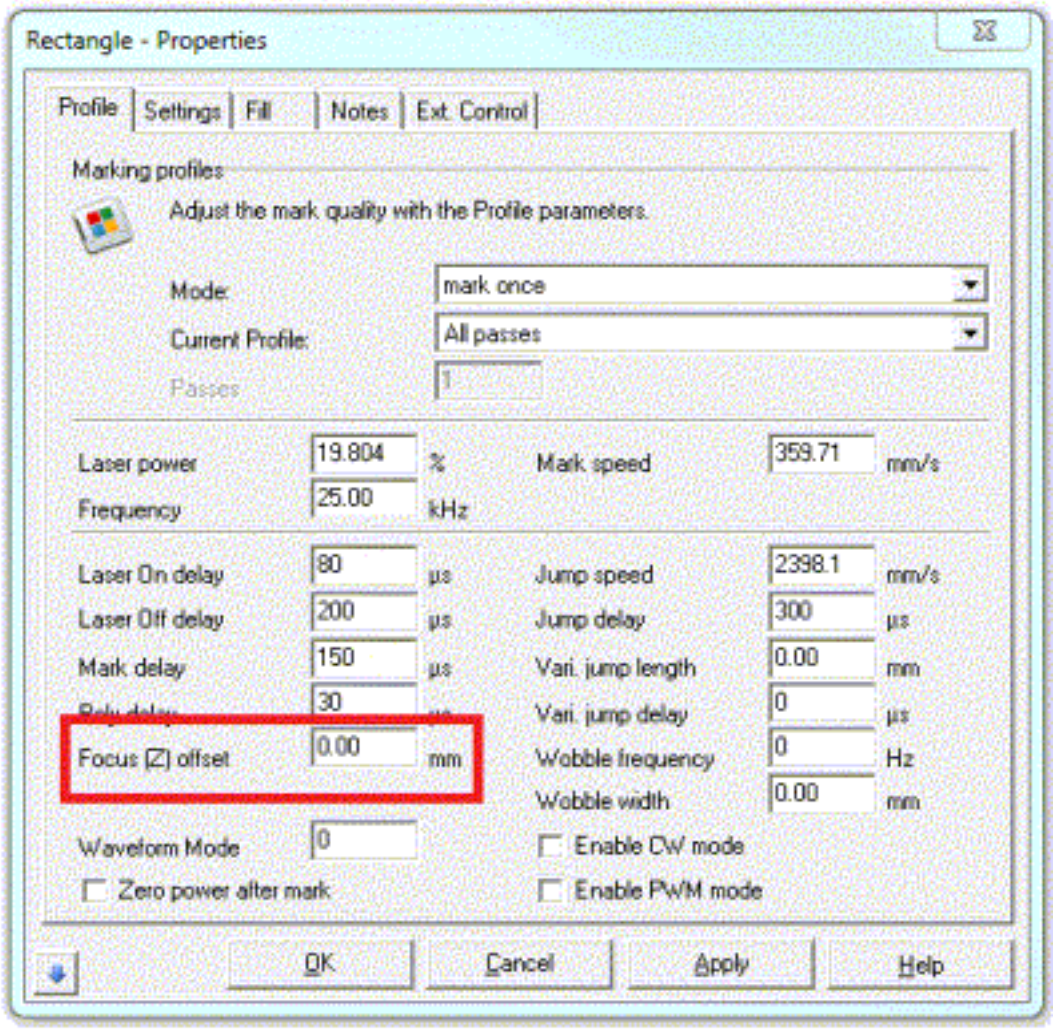
- Using multiple different fixtures or parts at different predetermined heights without moving the laser head. This simplifies tooling since the tooling doesn't need to conform to a regular 2D laser focal position (*Object Based Adjustment*)
- Marking on a part with multiple features or steps at different heights (*Object Based Adjustment)*
- Accommodating real-time changes in fixture position with vision system or part position sensor focus changes (*Autofocus Adjustments*)
- Setting up a new part or job of unknown height (*Autofocus Adjustments*)

Refer to *Appendix A* for details on the allowable span of adjustment in the Z-axis direction.

Object Based Adjustments

Object Based Adjustments are made in the WinLase software using the *Focus (Z) Offset* parameter and are applied each time an object is marked in a particular job. This is a WinLase object parameter and is referenced to the native Z = 0 position as shown on the previous page. It is necessary to use the minus sign (-) for positions Z < 0 (towards the Z Minimum direction). Each object can have an individual offset.

WinLase will use the sum of each object's individual *Focus (Z) Offset* parameter and any Autofocus offset applied to set the focal position at mark time of that specific object. This works in both streaming and standalone modes.



Autofocus Adjustments

Autofocus Adjustments are made using the remote API or HOST interface of the laser. They allow a global focus shift to be applied to all objects. The sum of the Autofocus shift and Object shift must be between $Z_{MIN} \leq Z \leq Z_{MAX}$ so that the laser mark can occur in the working envelope of the –AF head.

Syntax for an Autofocus Adjustment can be found in Appendix D of this manual or the appropriate WinLase manuals found in C:\marker\documentation. The Autofocus adjustment must occur over the API in standalone mode and HOST interface in streaming mode and can be made at any time in either mode.

The offset applied can come from a laser sensor, vision sensor, PLC, or any other device capable of taking a measurement and sending to the laser and is measured in microns with respect to Z=0.

The Autofocus Adjustment must be applied before the job is started so that the focus shift calculations have time to occur. In an automation environment with a sensor

Syntax –

Standalone Mode via Remote API

Command:	SetZOffsetRWU
Implementation:	"76,zoffest", where *zoffset* (in µm) is the z-axis focus change from Z=0. - The offset is absolute and can be positive or negative. - It must be between $Z_{MIN} \leq Z \leq Z_{MAX}$
Ex:	"76,-1000" offsets 1mm down
Response:	An API response. "0"=Acknowledged, "9"=Bad Argument, etc.

See Appendix D for the full list of API response codes.

Streaming Mode via Host Interface

Command:	Modify Zoffset
Implementation:	"MODIFY,zoffset,value", where *value* is in microns referenced to Z=0. - The offset is absolute and can be positive or negative. - It must be between $Z_{MIN} \leq Z \leq Z_{MAX}$
Ex:	"MODIFY,zoffset,-1000" offsets 1mm down
Response:	A HOST mode response.

To Reset:

Send command with "0" to rest to Z=0 microns

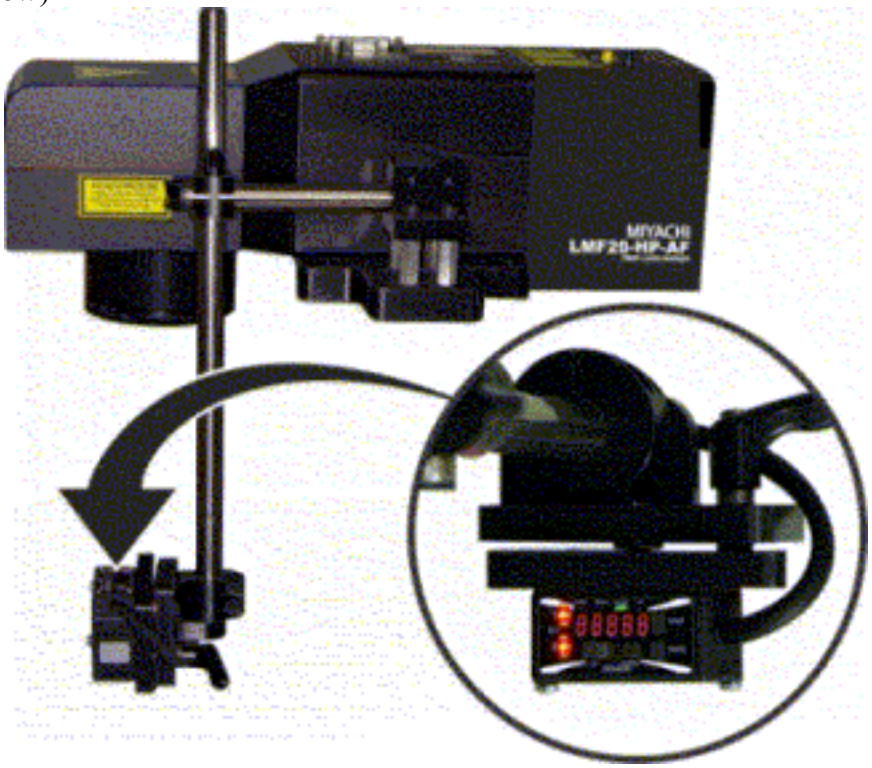
Section II: Configuring for and using your -AF marker with Autofocus using the 8-921-xx Autofocus system

For convenience, Amada Miyachi America has made available a laser distance sensor based Autofocus kit that interfaces with laser markers equipped with the –AF adjustable focus head. The autofocus uses a laser distance sensor to register the distance between the known Z=0 laser focal plane and the user part. This measured distance is sent to the laser marker and the laser focus is adjusted so that the laser beam is focused on the part. It is the user's responsibility to make sure that the sensor is aimed at the correct location and that the Z=0 position of the scan head and laser sensor are physically aligned.

The Autofocus kit contains the following components:

1. An Omron ZX-1 or equivalent laser sensor wired appropriately for the interface box.
    a. 100mm focal length, Amada Miyachi America Part Number 4-70245-01
    b. 300mm focal length, Amada Miyachi America Part Number 4-70245-02
2. Autofocus Interface Controller, Amada Miyachi America Part Number 4-70256-01
    This interface controller translates the raw sensor output into API or HOST mode commands to apply the measured offset to the laser focus offset. In addition, the interface controller is the place where focus related I/O takes place including triggering a reading to be implemented by the laser.
3. RS-232 cable between the Autofocus Interface Controller and the laser API or PC COM port.
4. Autofocus Interface Controller power cable, Amada Miyachi America Part Number 4-70245-01

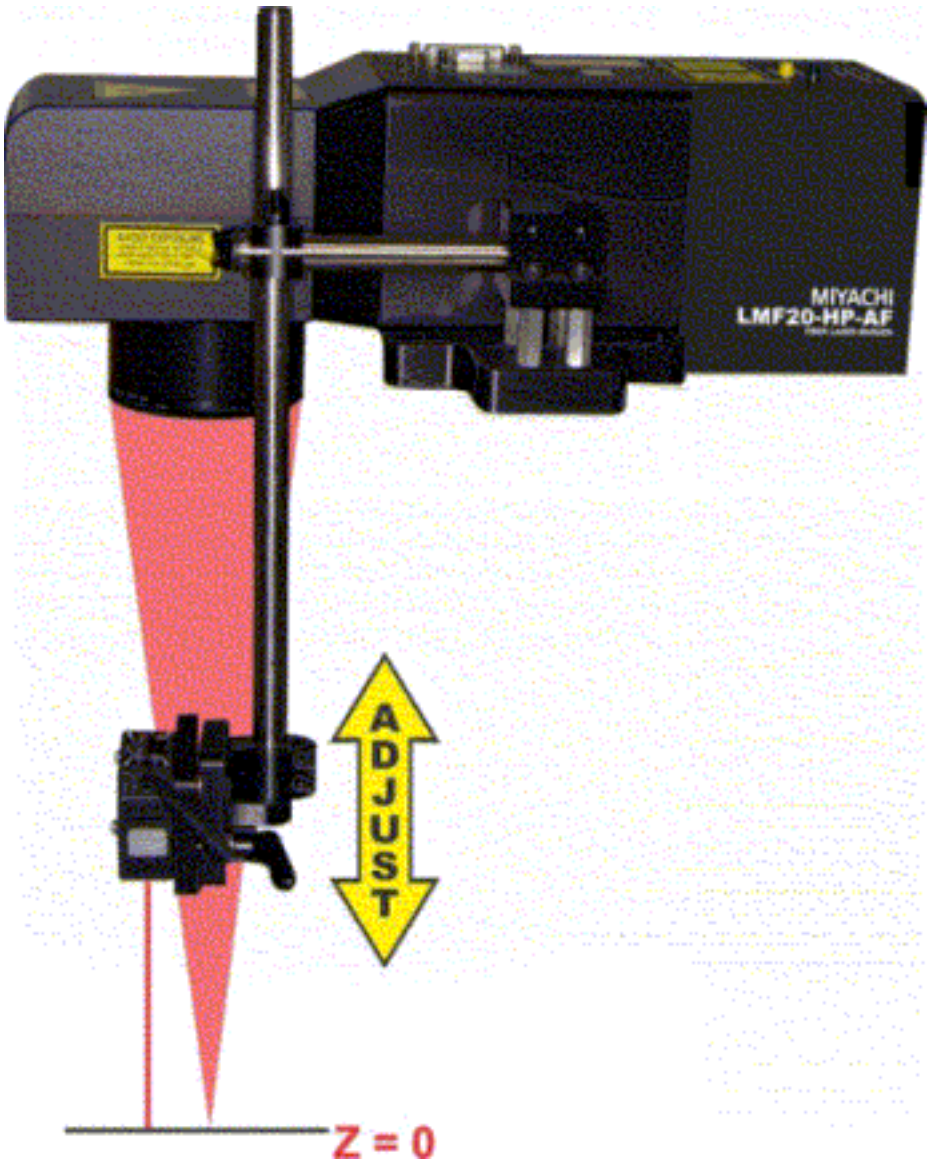
In addition, an optional mounting bracket system is available, Amada Miyachi America Part Number 8-922-01 (*shown below*)



Configuration

Before using the Autofocus system the user must determine whether or not the process will be executed in streaming or standalone/local mode. The COM cable from the Autofocus Interface Controller must be connected to either the PC's COM port for streaming mode or the laser's API COM port for standalone mode.

Note: Do not use the laser distance sensor's "Zero" function as it does not update the detected sensor value used by the Autofocus Interface Controller.



Standalone Mode Configuration

1. Connect the Autofocus Interface Controller to COM1 or COM2 of the laser rear panel using a RS-232 cable.
2. Connect the Autofocus Interface Controller power to the laser marker.
3. Connect a PLC or similar I/O bit to the Trigger input of the Autofocus Interface Controller.
4. In WinLase right click on the laser in the Laser System Viewer and click “Settings”.
5. Navigate to the COM Ports tab.
6. Set COM2 to Remote API and COM2 baud rate to 9600.
7. Click “OK”.
8. Put a piece of test material at the Z=0 position of the laser marker.
9. Mark a test mark with the Focus (Z) Offset parameter in WinLase set to 0.
10. Physically adjust the laser distance sensor until the display reads zero.
11. Run a standalone job at Z=0.
12. Move the test material to some other Z position, send two Autofocus trigger events, and trigger the mark. The Autofocus Interface Controller may require a second trigger the first time it is used to automatically select mode between API and HOST commands.
13. Verify that the mark is in focus
14. If the mark is out of focus verify the Autofocus Interface Controller scaling is set appropriately using the instructions later in this chapter.
15. From this point forward the Autofocus Interface controller requires a single trigger before a cycle start to update focus position.
16. If the laser distance sensor is out of range a red LED will illuminate and the fault bit on the Autofocus Interface Controller will be active.

Configuring Autofocus in the WinLase GUI in streaming mode

1. Connect the Autofocus Interface Controller to the PC COM port a RS-232 cable.
2. Connect the Autofocus Interface Controller power to the laser marker.
3. Connect a PLC or similar I/O bit to the Trigger input of the Autofocus Interface Controller
4. In WinLase click on System from the dropdown menu at the top, then Preferences
5. Click the "COM Ports" tab
6. Click on "Host Interface"
7. Click "Make port available to Host Interface"
8. Use the "Port #" dropdown to select the COM port to which the Autofocus Interface Controller is connected
9. Set Baud 9600, Data Bits 8, Stop Bits 1, Parity None and click "Apply"
10. Click the "Host Interface" tab
11. Change "Host Connection" to "RS-232"
12. Click OK
13. Put a piece of test material at the Z=0 position of the laser marker.
14. Mark a test mark with the Focus (Z) Offset parameter in WinLase set to 0
15. Physically adjust the laser distance sensor until the display reads zero
16. Run a job using "Run"
17. Move the test material to some other Z position, send two Autofocus trigger events, and again Run the mark. The Autofocus Interface Controller may require a second trigger the first time it is used to automatically select mode between API and HOST commands.
18. Verify that the mark is in focus
19. If the mark is out of focus verify the Autofocus Interface Controller scaling is set appropriately using the instructions later in this chapter.
20. From this point forward the Autofocus Interface controller requires a single trigger before a cycle start to update focus position.
21. If the laser distance sensor is out of range a red LED will illuminate and the fault bit on the Autofocus Interface Controller will be active.

Using Autofocus

Once configured correctly using Autofocus is easy. Before running a job that needs an Autofocus just send a Trigger I/O bit to the Autofocus Interface Controller. Although the total speed of the Autofocus trigger event is variable depending on RS-232 timing it is usually completed in < 150ms.

Section III: Autofocus Interface Controller Technical Information

The Autofocus Interface Controller is a small box connected between the laser marker and the laser distance sensor. Its primary purpose is to sample the laser distance sensor analog reading on command, convert the reading into a real world unit value, and send via RS-232 to the fiber laser marker.

The Autofocus Interface Controller has no user interface or monitoring features beyond what is available over I/O, RS-232, or on the sensor itself. Autofocus Interface Controller inputs and outputs are equivalent in specification to fiber laser marker inputs and outputs. The Autofocus Interface Controller is powered by the fiber laser marker and provides power to the laser distance sensor.

A Microchip PIC16F1788 is used in the Autofocus Interface Controller to process I/O and communicate with the sensor and laser marker.

Inputs (by pin number):

1. **TRIGGER**

   This input triggers a reading of the laser distance sensor. The autofocus interface controller samples the 4-20mA current output of the sensor, converts to a real world unit measurement in microns, and then sends the measurement to the laser marker.

2. **TUNE1**

   Allows the user to trigger the TUNE1 input of the laser distance sensor to trigger tuning of the sensor's OUT1 output.

3. **TUNE2**

   Allows the user to trigger the TUNE2 input of the laser distance sensor to trigger tuning of the sensor's OUT2 output.

4. **ZERO**

   DO NOT USE – Not compatible with Autofocus operation.

5. **LD_OFF**

   This input is used to turn off the laser diode in the laser distance sensor. The laser distance sensor's LD Off input is also controlled by the fiber laser marker interlock circuit. The PIC microcontroller uses "OR" logic to turn off the laser diode when either the LD_OFF input or the laser marker interlock is activated.

6. **+24V**

   24V power supply for input bias or triggering inputs depending on mode of integration.

7. **INPUT_BIAS**

   This input is used to set the optocoupler bias to determine the active state of input signals.

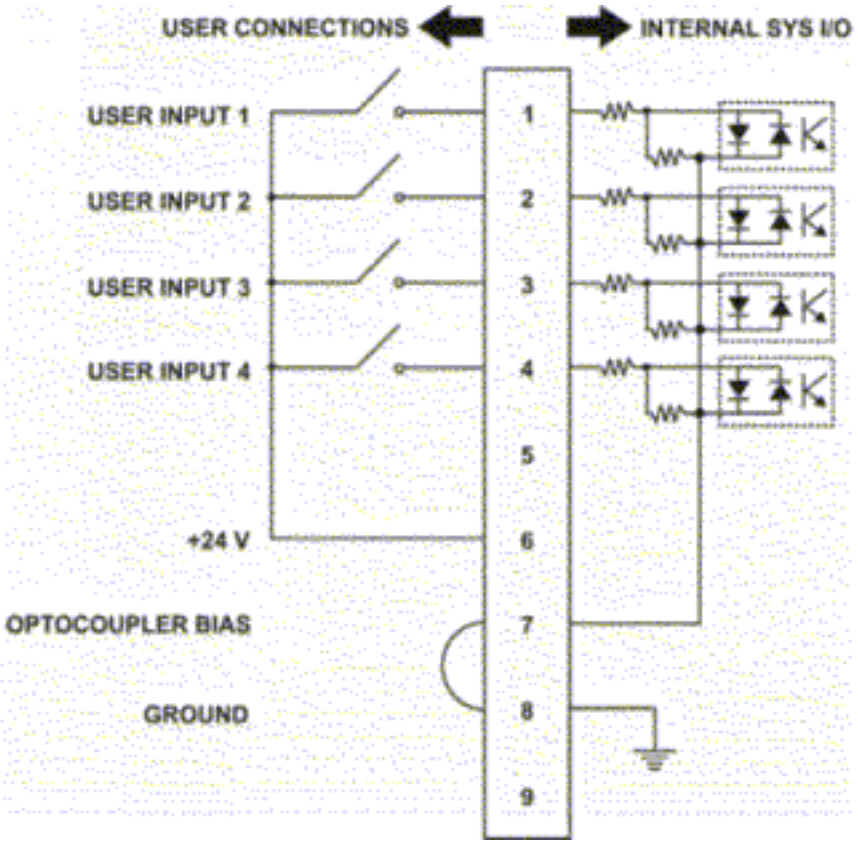
8. **GND**

   I/O Ground can be used to set input bias or trigger inputs depending on mode of integration.

Sample Input User Wiring Schematic

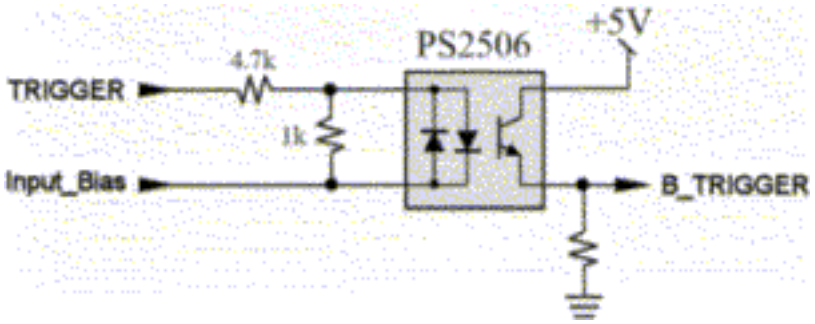
User I/O IN – Dry contact Input Switches (internally biased)

Note: Pin definitions as defined above.



Example of Input Protection Schematic

The inputs accept 24V inputs in a sourcing or sinking configuration depending on the wiring of Input_Bias. All outputs are fully isolated using PS2506L-1-A optoisolators. A sample schematic is shown below for the TRIGGER input. In the sample schematic TRIGGER comes directly from the I/O input and B_TRIGGER is connected to the PIC microcontroller.

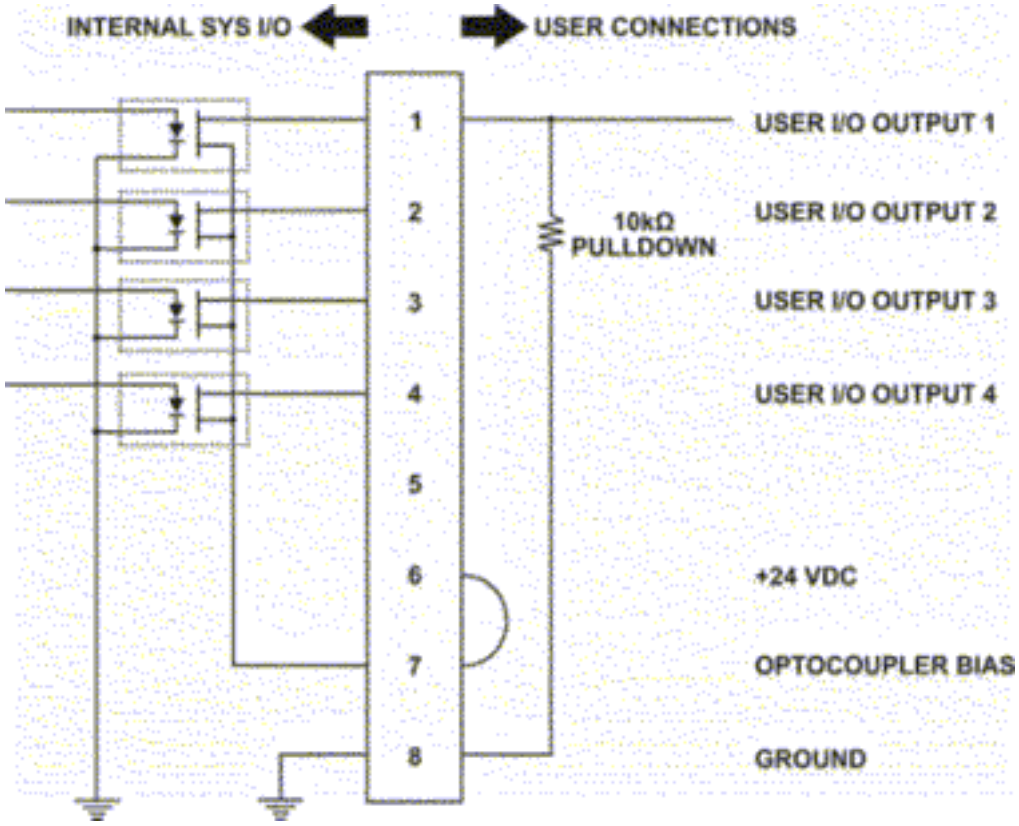


Outputs (by pin number)

1. **OUT1**
   This output allows the user to monitor the laser distance sensor's OUT1 output.
2. **OUT2**
   This output allows the user to monitor the laser distance sensor's OUT2 output.
3. **ERROR**
   This output allows the user to monitor the autofocus interface controller Error bit which indicates various errors including sensor out of range, fiber marker offset value out of range, and other cases. See the error definition section of this document.
4. **24V**
   24V power input to the autofocus interface controller. This must be connected to a 24V output on the back of the fiber laser marker.
5. **GND**
   Ground connection to autofocus interface controller. This must be connected to a Ground pin on a fiber laser marker I/O header. This is the primary ground between the fiber laser marker and the autofocus interface controller.
6. **24V**
   24V power supply for output biasing as needed
7. **OUTPUT_BIAS**
   Optocoupler biasing for outputs to determine active state of output signals
8. **GND**
   Ground for output biasing as needed

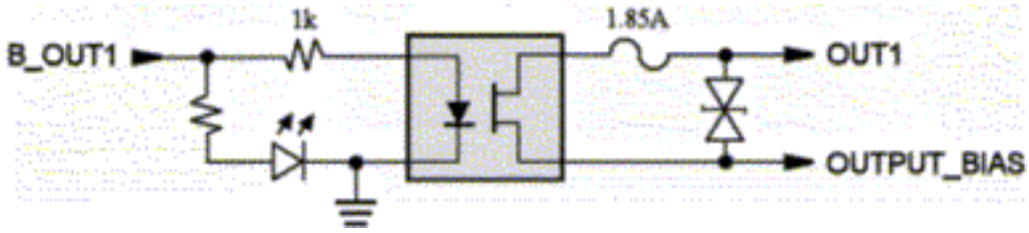
Sample Output User Wiring Schematic – Sourcing, with pull-down resistor

Note: Pin Definitions as described above



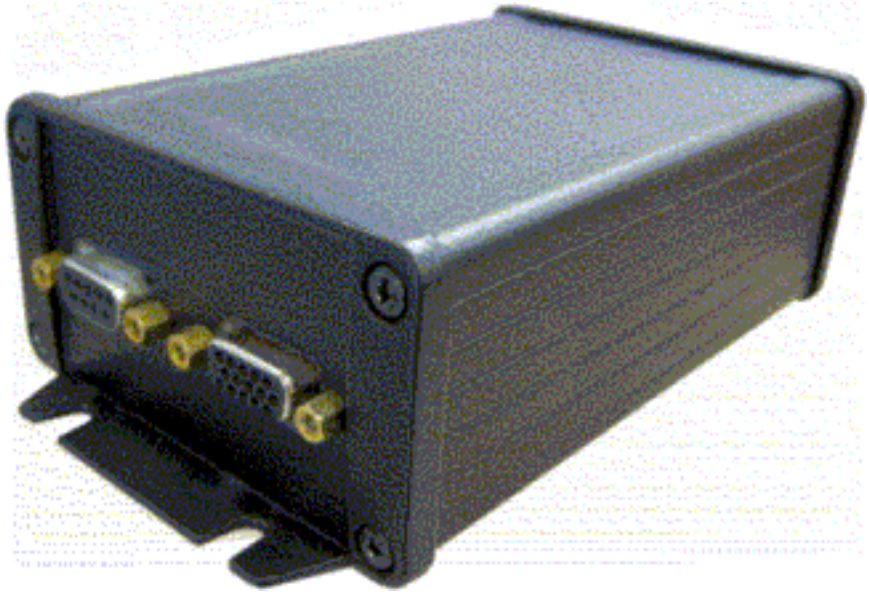
Example of Output Protection Schematic

The outputs can source or sink depending on the wiring of OUTPUT_BIAS. All outputs are optical solid state relays protected by 1.85A poly fuses, allowing a specified current draw of 500mA. Transient Voltage Suppressors are installed to deal with spikes. The sample schematic below shows the circuitry protecting OUT1. The B_OUT1 signal comes from the PIC, through the optical isolator, and to the output on the side of the autofocus interface.



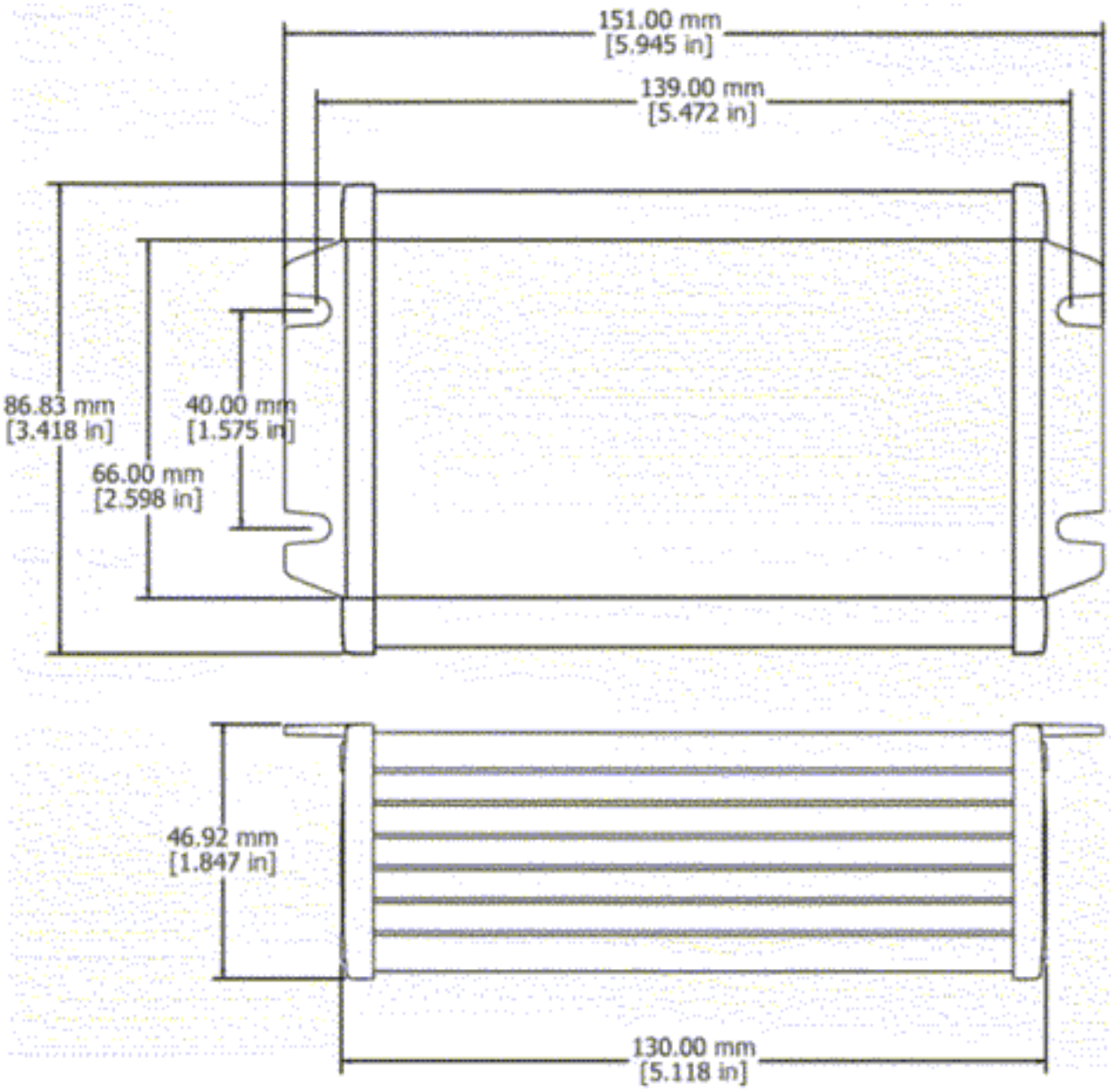
Autofocus Interface Controller Physical Attributes

The 4-70256-01 Autofocus Interface Controller is a small inline box with mounting flanges. On one end it has a female DB9 connection to the fiber laser marker COM port as well as a female HD15 connection to the laser distance sensor.

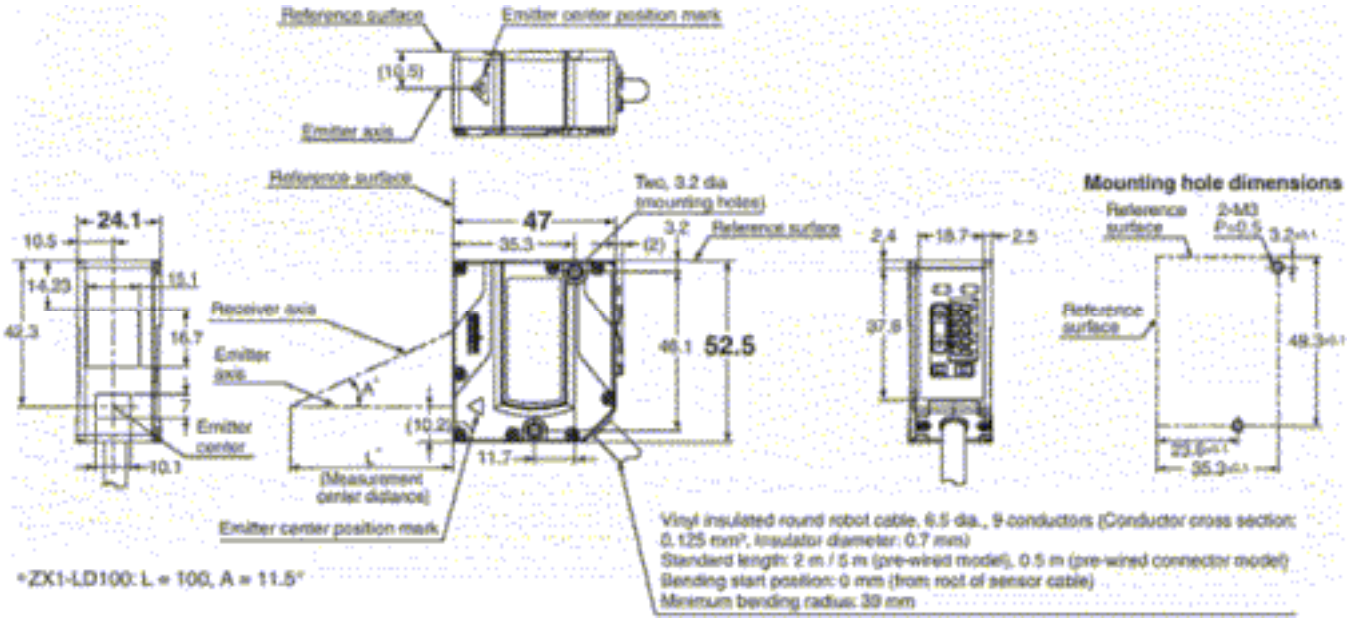


The other end has two rows of connections for Phoenix 3.81mm screw type terminal blocks. The first row (A1-8) is inputs and the second row (B1-8) is outputs. Two LEDs (GREEN = Power, RED = Fault) are present next to the terminal connections but not shown in these photos.

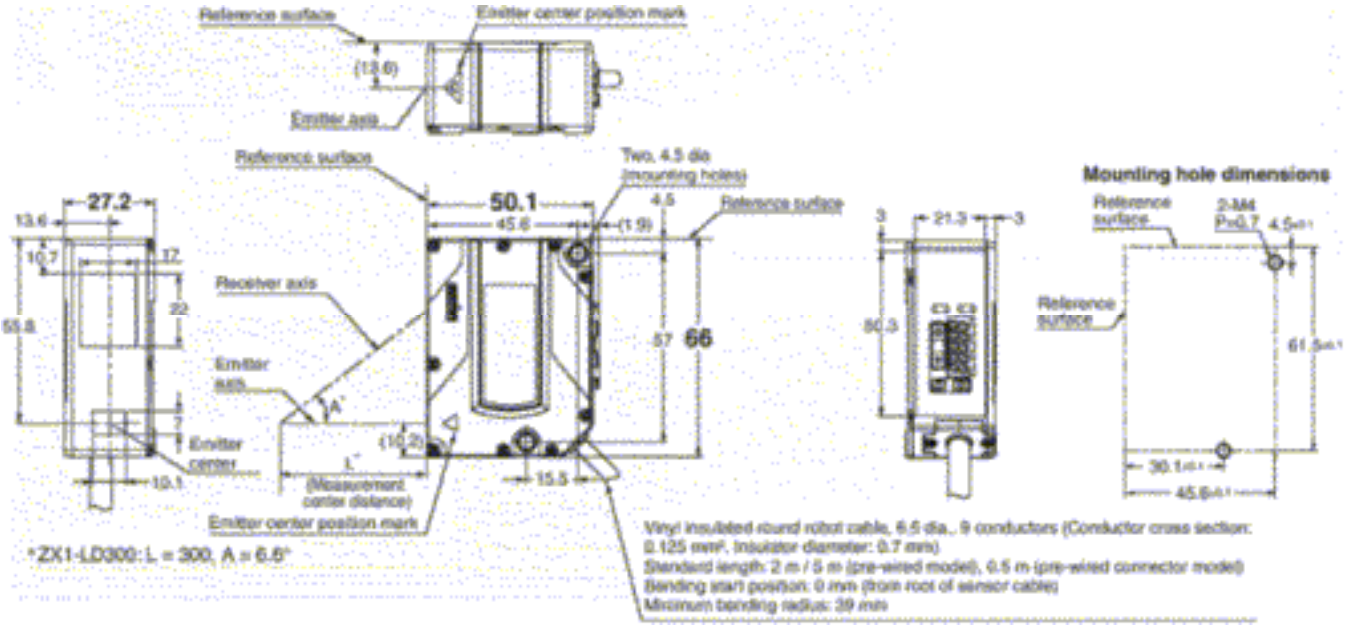
Autofocus Interface Controller Dimensions (Connectors not shown)



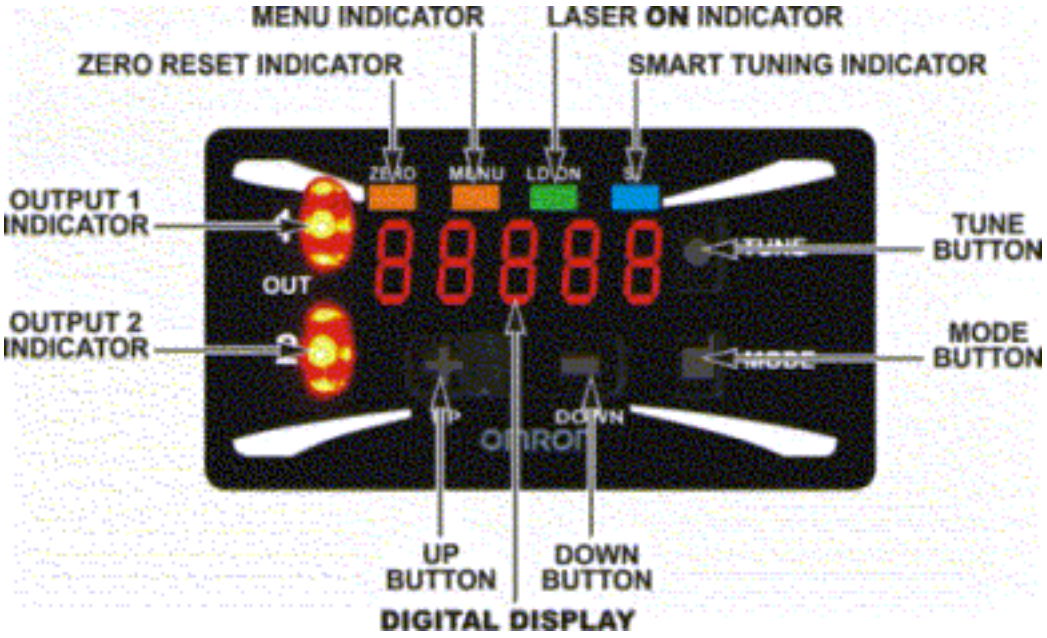
ZX1-LD100A Drawing – Amada Miyachi America Part Number 4-70245-01



ZX1-LD300A Drawing – Amada Miyachi America Part Number 4-70245-02



Displays, Indicators, and Controls



Sensor Manufacturer Specifications

Item		4-70245-01	4-70245-02
Model		ZX1-LD100A61	ZX1-LD300A61
Output		NPN Output	NPN Output
Measurement Range		100 ± 35mm	300 ± 150mm
Light Source (wavelength)		Visible-light semiconductor laser (660nm, 1mW max., IEC/EN Class 2, FDA Class 2)*1	
Spot Diameter (typical) *2		0.33mm diameter	0.52mm diameter
Power Consumption		2,500mW max. (105mA max. at 24VDC, 210mA max. at 12VDC)	
Current Consumption		250mA max. (at power supply voltage 10VDC)	
Control Output		Load power supply voltage: 30VDC max., Load current: 100mA max. [Residual voltage: 1V max., (≤ 10mA load current), 2V max., (10mA – 100mA load current)]	
Analog Output		Current output: 4 to 20mA, maximum load resistance: 300Ω	
Functions		Smart tuning, keep function, background removal, OFF-delay timer, ON-delay timer, zero reset, area output, eco function, hysteresis width setting, and setting initialization	
Indicators		Digital display (red), output indicator (OUT1, OUT2) (orange), zero reset indicator (orange), menu indicator (orange), laser ON indicator (green), and smart tuning indicator (blue)	
Response Time	Judgment Output	Super-High-Speed (SHS) Mode: 1mS High-Speed (HS) Mode: 10mS Standard (Std) Mode: 100mS	
	Laser OFF input	200mS max.	
	Zero reset input	200mS max.	
Temperature Characteristics *3		0.03% F.S./ºC	
Linearity *4		± 0.15% F.S.	± 0.25% F.S.
Resolution *5		7µm	30µm
Ambient Illumination *6		7,500 lx or less (incandescent)	5,000 lx or less (incandescent)
Ambient Temperature (non-condensing)		Operating: -10 to +55ºC, Storage -15 to +70ºC	
Ambient Humidity (non-condensing)		Operating and Storage: 35% to 85%	
Dielectric Strength		1,000 VAC, 50/60Hz, 1 minute	
Vibration Resistance (destruction)		10 to 55Hz, 1.5mm double amplitude, 2 hrs. each (X,Y & Z directions)	
Shock Resistance (destruction)		500 m/s^2 3 times each in X, Y and Z directions	
Degree of Protection *7		IEC 60529, IP67	
Connection Method		Pre-wired model (standard cable length: 2m)	
Weight (packed state / sensor only)		Approx. 240g / Approx. 180g	Approx. 270g / Approx. 210g

*1 Classified as Class 2 by IEC60825-1 criteria in accordance with the FDA standard provisions of Laser Notice No. 50 CDRH registration has been completed. (Center for Devices and Radiological Health, Accession Number: 1210041)

*2 Spot diameter is defined at the measurement center distance and is defined as 1/e^2 (13.5%) of the central intensity at the measurement center distance. False detections can occur in the case there is light leakage outside the defined region and the surroundings of the target object have a high reflectance in comparison to the target object. Accurate measurements may not be possible for workpieces that are smaller than the spot diameter.

*3 Temperature characteristics: Value for the case the space between the sensor and the standard target object is secured by an aluminum jig. (Measured at the measurement center distance).

*4 Linearity: Indicates the error with respect to the ideal straight line of the displacement output in the case of measuring the standard target object (white ceramic) at a temperature of 25ºC

*5 Resolution: Defined in Standard Mode for the standard target object (white ceramic) after executing Smart Tuning. The resolution indicates the repetition accuracy for a still workpiece. Not an indication of the distance accuracy. Resolution performance may not be satisfied in a strong electromagnetic field.

*6 The Ambient Illumination is the illumination on the received light surface.

*7 IP67 protection applies to the connector on pre-wired models.

Section IV: Calibrating the Autofocus Interface Controller to a Laser Sensor

The Autofocus Interface Controller is calibrated as shipped – if the sensor changes it may be necessary to repeat calibration.

EQUIPMENT REQUIRED:

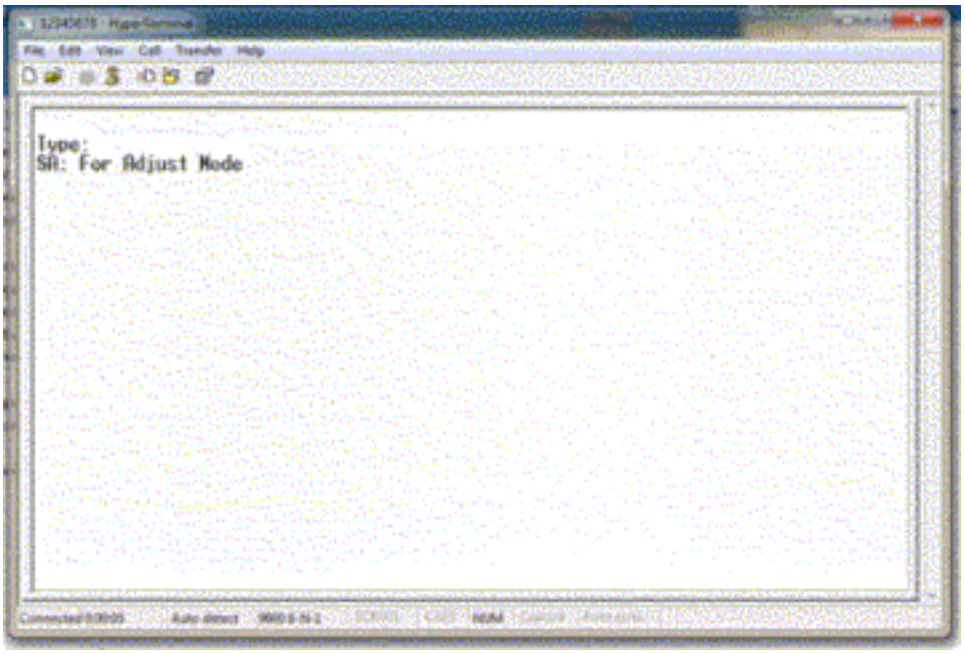
1. 8-921-01 or 8-921-02 Autofocus accessory kit.
2. PC equipped with a RS-232 port and terminal software.
3. Straight through RS-232 cable.
4. Adjustable fixture to mount laser sensor 100mm or 300mm away from nominal target for calibration purposes.
5. Laser to provide power.
6. Autofocus Interface Controller power cable.

SETUP:

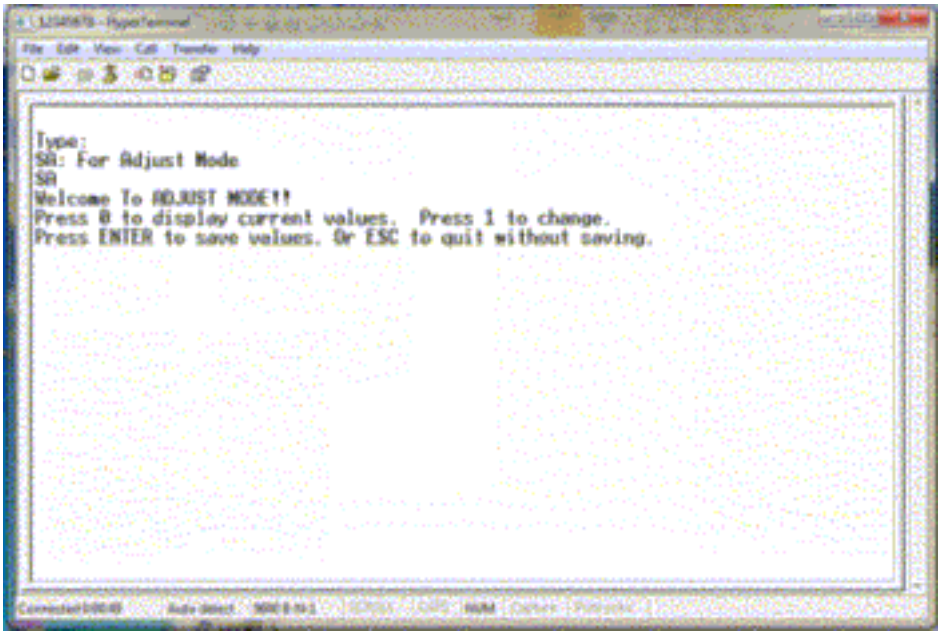
1. Mount laser sensor approximately 100mm (for 8-921-01) or 300mm (for 8-921-02) away from target.
2. Plug laser sensor HD25 plug into interface box.
3. Use RS-232 cable to connect interface box to PC serial port.
4. Open terminal program and establish a connection at 9600 Baud, 8-N-1-None (no parity).
5. Connect test fixture I/O input and output harnesses.
6. Plug in autofocus interface controller power cable and turn on laser to provide power.

PROCEDURE:

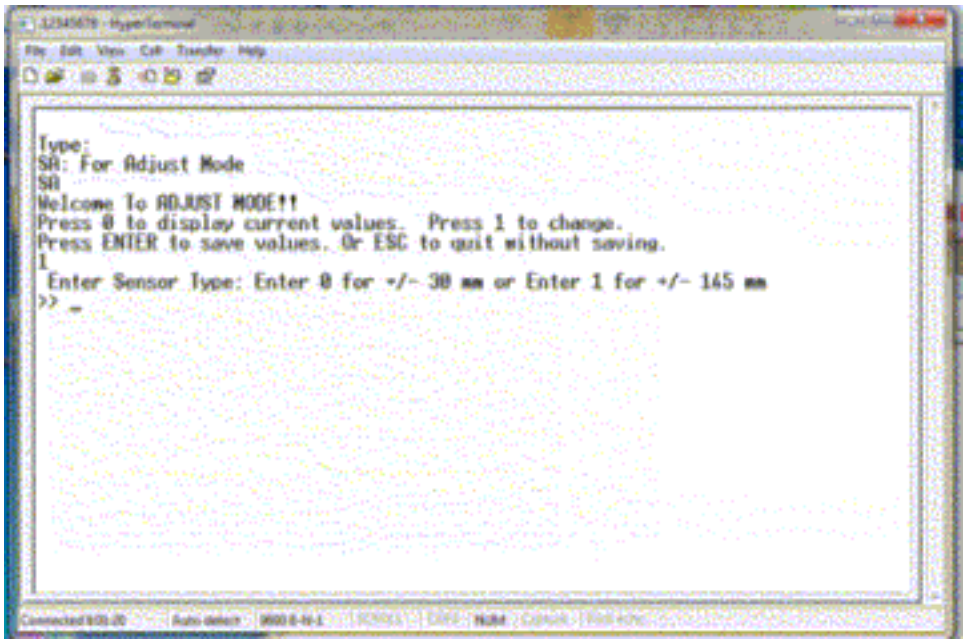
1. Use the terminal application to create a connection. Press carriage return if nothing shows on the screen. The menu should come up as shown below:



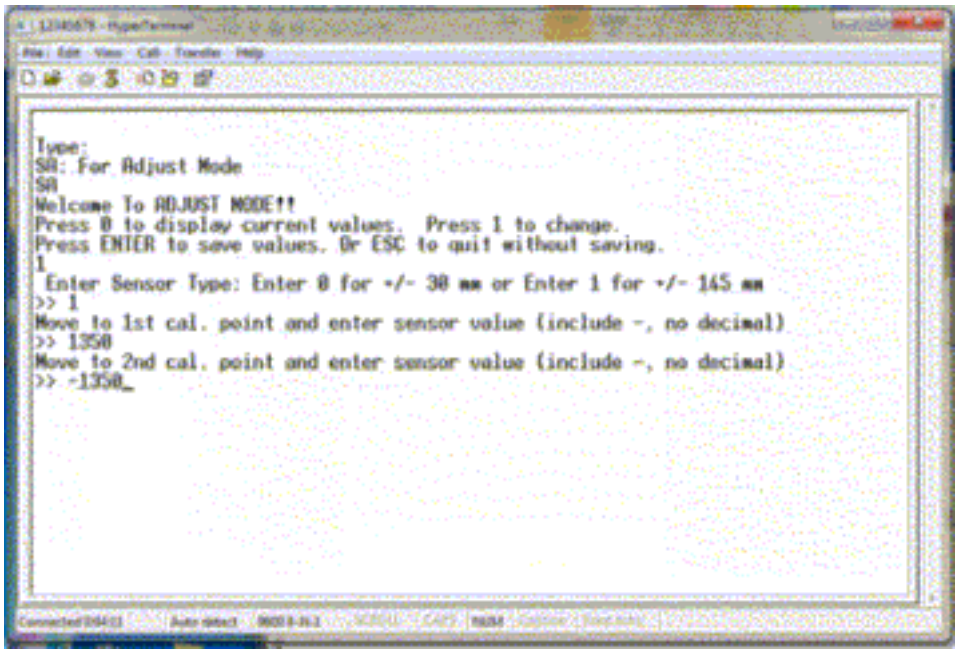
2. Type “SA” and press enter to enter adjust mode



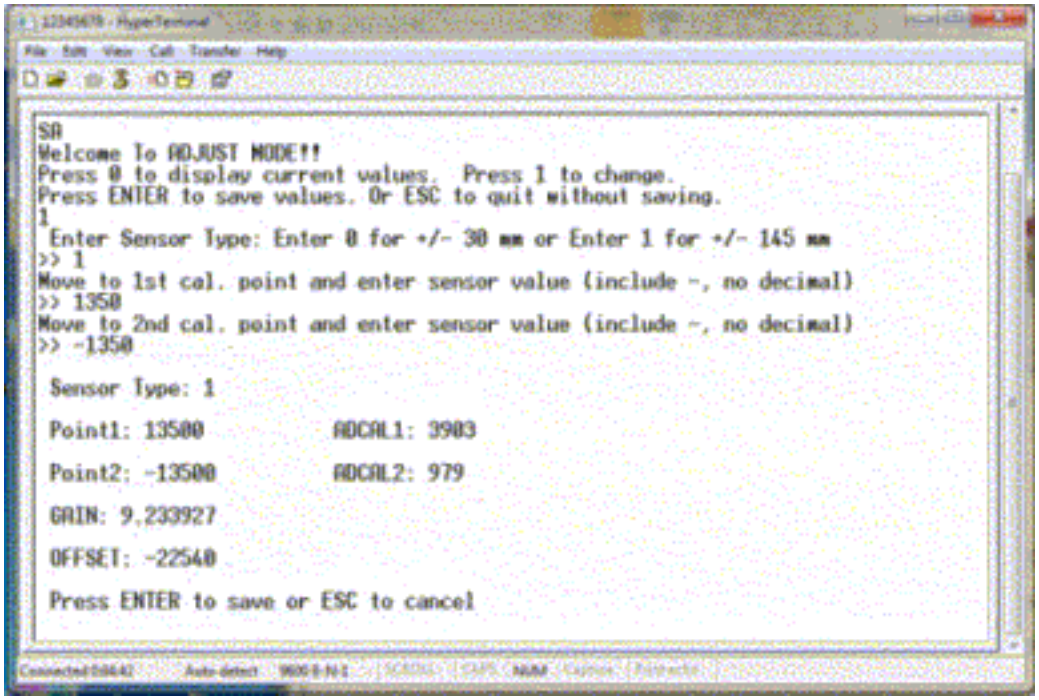
3. Send a “1” to set analog calibration values



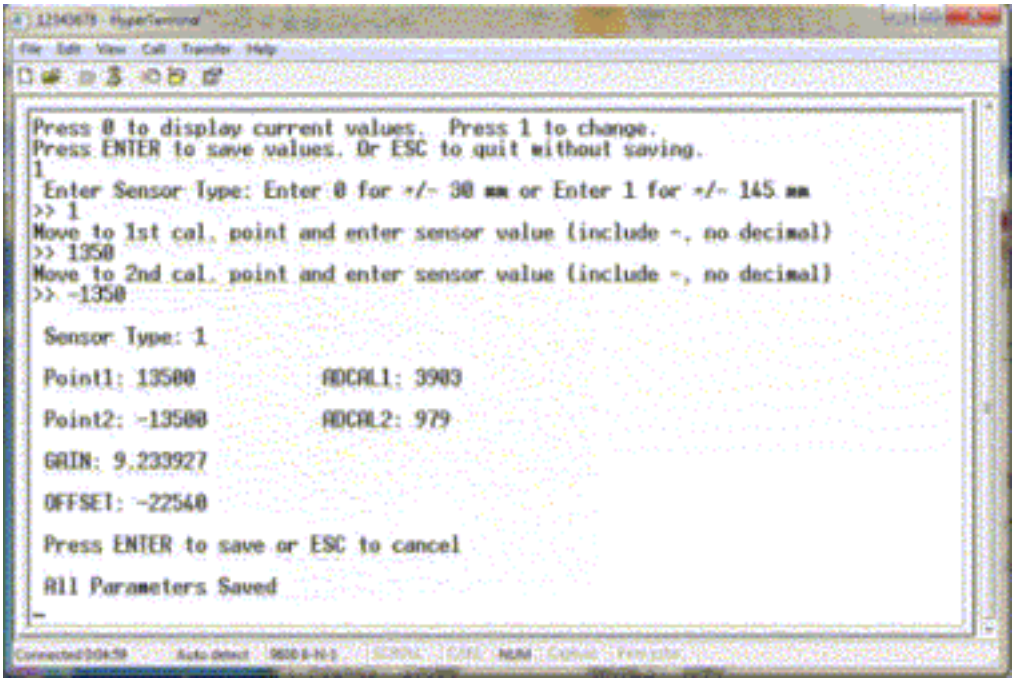
4. Select the sensor type using 0 or 1.
5. Move the table in one direction near the maximum value and type in the value. Repeat in the opposite direction. For the examples shown 135.0mm and -135.0mm were selected. Do not enter decimals, just the sign and numerals shown on the sensor display.



6. After entering the second value the calculated values will be displayed. Press Enter to save as prompted on the display



The system will indicate that the data is saved



Appendix A
Technical Specifications

Section I. Laser Specifications

SPECIFICATIONS – LMF50, LMF20, LMF10					
Parameter			**LMF50**	**LMF20**	**LMF10**
			8-79-Bxx-xAA	**8-79-Cxx-xAA**	**8-79-Dxx-xAA**
Laser	Oscillator Wavelength		1060-1150nm, central emission at 1064 ± 5nm		
	Oscillation Mode		TEM00-Mode (at or near)		
	M^2		≤ 2, typical < 1.4	≤ 2, typical ~ 1.5	
	Minimum Continuous Laser Power from Engine		> 48.0W @ 50-200kHz	> 19.0W @ 20-200kHz	> 9.5W @ 20-200kHz
	Minimum Continuous Output Power at Aperture		> 43.0W @ 50-200kHz	> 17.1W @ 20-200kHz	> 8.5W @ 20-200kHz
	Type of Oscillation		Q-Switched		
	Pulse Frequency		50-200kHz	20-200kHz	
	# unique Pulse Tune Waveforms		N/A		
	Guide Beam Wavelength		660nm (0.2mW at output)		
Power Supply	Power Supply		90-130VAC, 180-260VAC ± 10%, 50/60Hz, Single-Phase		
	Maximum Running Current		7A @ 110VAC		
	Recommended AC Service		10A @ 110VAC		
Environmental	Ambient Operating Temperature		41-95°F (5-35°C)		
	Relative Humidity		Less than 90% (non-condensing)		
	Head Sealing		IP-54 rated, higher with protection of lens aperture		
	Installation Site		Do not use where there is considerable dirt, dust, oil mist, chemicals, fumes, moisture, vibration or near a high-frequency noise source.		
Physical Properties	Mass	Control Unit	61 lbs. (27.7kg)		
		Head	8.4 lbs. (3.8kg)		
	Dimensions (W x H x D)	Control Unit	17.0" x 7.83" x 22.21" (431.8mm x 198.9mm x 564.1mm)		
		Head	3.03" x 4.22" x 16.01" (77mm x 107.2mm x 406.6mm)		

SPECIFICATIONS – LMF70-HP, LMF35-HP				
Parameter			**LMF70-HP**	**LM35-HP**
			8-79-Exx-xxA	**8-79-Qxx-xxA**
Laser	Oscillator Wavelength		1060-1150nm, central emission at 1064 ± 5nm	
	Oscillation Mode		TEM00-Mode (at or near)	
	M^2		< 1.6	2.5 ≤ M^2 ≤ 3.5
	Minimum Continuous Laser Power from Engine[1]		71.0W ≤ P_O ≤ 73.0W @ 70kHz, wf0	> 38.0W @ 30-500kHz
	Minimum Continuous Output Power at Aperture		> 63.5W @ 70-500kHz	> 34.2W @ 30-500kHz
	Type of Oscillation		CW or Pulsed	
	Pulse Frequency		2-500kHz	
	# unique Pulse Tune Waveforms		36	23
	Guide Beam Wavelength		650nm (< 5mW at output)	
Power Supply	Power Supply		90-130VAC, 180-260VAC ± 10%, 50/60Hz, Single-Phase	
	Maximum Running Current		7A @ 110VAC	
	Recommended AC Service		10A @ 110VAC	
Environmental	Ambient Operating Temperature		41-95°F (5-35°C)	
	Relative Humidity		Less than 90% (non-condensing)	
	Head Sealing		IP-54 rated, higher with protection of lens aperture	
	Installation Site		Do not use where there is considerable dirt, dust, oil mist, chemicals, fumes, moisture, vibration or near a high-frequency noise source.	
Physical Properties	Mass	Control Unit	61 lbs. (27.7kg)	
		Head	8.4 lbs. (3.8kg)	
	Dimensions (W x H x D)	Control Unit	17.0" x 7.83" x 22.21" (431.8mm x 198.9mm x 564.1mm)	
		Head	3.03" x 4.22" x 16.95" (77mm x 107.2mm x 430.4mm)	

Note 1: Power Output is available from 2-500kHz. The power output between 2kHz and the minimum frequency (at rated power) will be limited. See *Appendix C* for more information.

SPECIFICATIONS – LMF20-SM, LMF20-HP				
Parameter			**LMF20-SM**	**LMF20-HP**
			8-79-Pxx-xxA	**8-79-Rxx-xxA**
Laser	Oscillator Wavelength		1060-1150nm, central emission at 1064 ± 5nm	
	Oscillation Mode		TEM00-Mode (at or near)	
	M^2		$\leq$ 1.3	1.6 $\leq$ M^2 $\leq$ 2.0
	Minimum Continuous Laser Power from Engine[1]		> 20.0W @ 40-500kHz	> 20.0W @ 25-500kHz
	Minimum Continuous Output Power at Aperture		> 17.9W @ 40-500kHz	> 17.9W @ 25-500kHz
	Type of Oscillation		CW or Pulsed	
	Pulse Frequency		2-500kHz	
	# unique Pulse Tune Waveforms		23	24
	Guide Beam Wavelength		650nm (< 5mW at output)	
Power Supply	Power Supply		90-130VAC, 180-260VAC ± 10%, 50/60Hz, Single-Phase	
	Maximum Running Current		7A @ 110VAC	
	Recommended AC Service		10A @ 110VAC	
Environmental	Ambient Operating Temperature		41-95°F (5-35°C)	
	Relative Humidity		Less than 90% (non-condensing)	
	Head Sealing		IP-54 rated, higher with protection of lens aperture	
	Installation Site		Do not use where there is considerable dirt, dust, oil mist, chemicals, fumes, moisture, vibration or near a high-frequency noise source.	
Physical Properties	Mass	Control Unit	61 lbs. (27.7kg)	
		Head	8.4 lbs. (3.8kg)	
	Dimensions (W x H x D)	Control Unit	17.0" x 7.83" x 22.21" (431.8mm x 198.9mm x 564.1mm)	
		Head	3.03" x 4.22" x 16.95" (77mm x 107.2mm x 430.4mm)	

Note 1: Power Output is available from 2-500kHz. The power output between 2kHz and the minimum frequency (at rated power) will be limited. See *Appendix C* for more information.

SAFETY CIRCUIT SPECIFICATIONS	
Parameter	**Specification**
Emergency Stop Circuit Mechanical Lifetime	10 Million Cycles
Interlock Circuit Mechanical Lifetime	10 Million Cycles
Safety Circuit Mechanical Lifetime	10 Million Cycles
t_M (in years)	20
PL in accordance with IEC 13849-1	PLe (Cat 4) if appropriately integrated by end user
Dual Channel E-Stop Output Monitoring Relay Maximum Electrical Characteristics	24VDC, 1A

Marking Area – 2D Head (8-79-xxR-xxx)

LMF Lens Marking Area – 2D Head		
***f* θ Lens Unit**	***f* = 100mm**	***f* = 160mm**
Scanning Method	Galvanometer Scanner	
λ (wavelength)	1060-1150, central emission 1064 ± 5 nm	
Marking Area	2.42 in. x 2.42in. (61.5mm x 61.5mm)	3.89 in. x 3.89 in. (98.9mm x 98.9mm)
Working Distance (approximate)	3.86 ± 0.04 in. (98 ± 1mm)	6.93 ± 0.08 in. (176 ± 2mm)
Position Resolution	0.00016 in. (4μm)	0.00028 in. (7μm)

LMF Lens Marking Area – 2D Head		
***f* θ Lens Unit**	***f* = 254mm**	***f* = 420mm[1]**
Scanning Method	Galvanometer Scanner	
λ (wavelength)	1060-1150, central emission 1064 ± 5 nm	
Marking Area	6.18 in. x 6.18 in. (157 mm x 157mm)	11.44 in. x 11.44 in. (290.6mm x 290.6mm)
Working Distance (approximate)	11.65 ± 0.12 in. (296 ± 3 mm)	19.45 ± 0.20 in (494 ± 5 mm)
Position Resolution	0.00044 in. (11μm)	0.00088 in. (22μm)

Marking Area – AF (Adjustable Focus) Head (8-79-xxB-xxx)

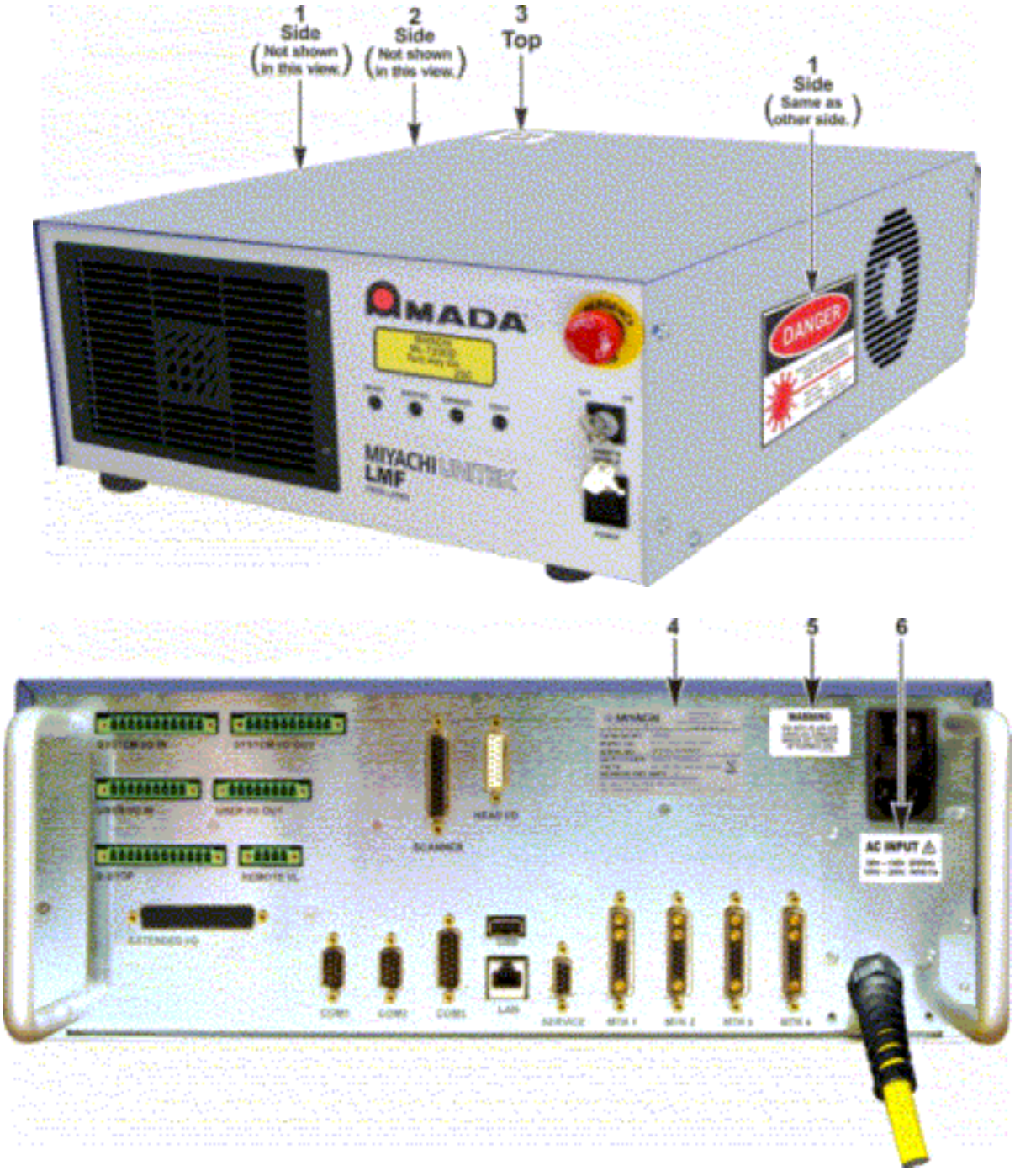
LMF Lens Marking Area – AF Head		
f **θ Lens Unit**	***f*** **= 100mm**	***f*** **= 160mm**
Scanning Method	Galvanometer Scanner	
λ (wavelength)	1060-1150, central emission 1064 ± 5 nm	
Marking Area	1.42 in. x 1.42 in. (36mm x 36mm)	2.99 in. x 2.99 in. (76mm x 76mm)
Working Distance (approximate)	3.84 ± 0.04 in. (97.5 ± 1mm)	6.81 ± 0.08 in. (172.9 ± 2mm)
Z-Axis Adjustment	± 0.197 in. (± 5mm)	± 0.472 in. (±12mm)

LMF Lens Marking Area – AF Head	
f **θ Lens Unit**	***f*** **= 254mm**
Scanning Method	Galvanometer Scanner
λ (wavelength)	1060-1150, central emission 1064 ± 5 nm
Marking Area	5.59 in. x 5.59 in. (142 mm x 142mm)
Working Distance (approximate)	11.65 ± 0.12 in. (295.9 ± 3 mm)
Z-Axis Adjustment	± 1.181 in. (± 30mm)

Note: *f*420mm f-theta lens not compatible with the AF head (8-79-xxB-xxx)

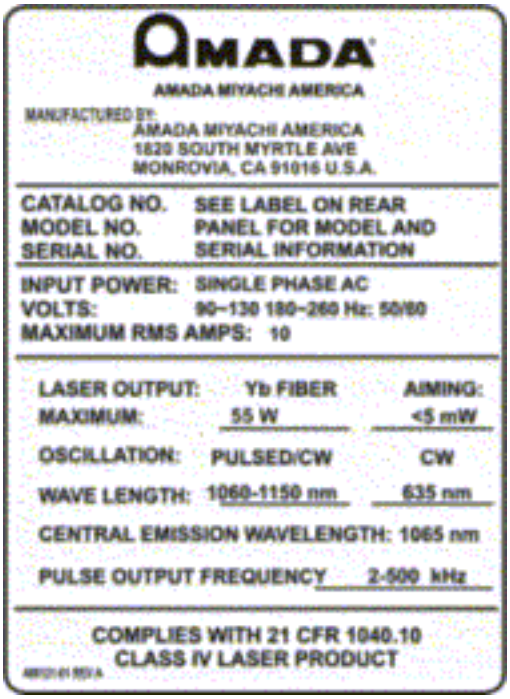
Section II. Warning and Identification Labels

Location on Equipment




DANGER
VISIBLE AND/OR INVISIBLE LASER RADIATION
AVOID EYE OR SKIN EXPOSURE TO
DIRECT OR SCATTERED RADIATION
MAX OUTPUT 55W /5mW
PULSE DURATION 10 - 190 ns
WAVELENGTH(S) 1050 -1200nm / 650 nm
CLASS IV LASER PRODUCT

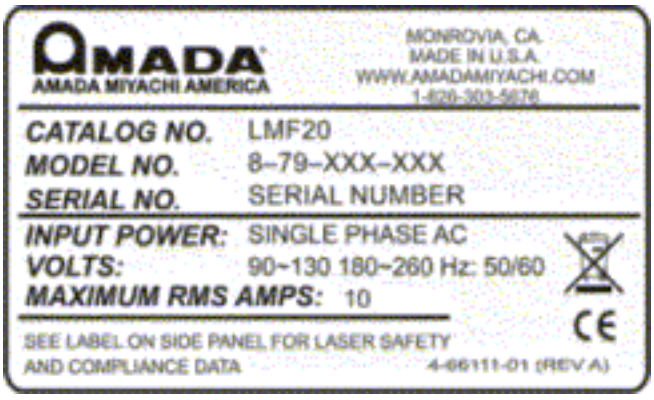
1


AMADA
AMADA MIYACHI AMERICA
MANUFACTURED BY:
AMADA MIYACHI AMERICA
1820 SOUTH MYRTLE AVE
MONROVIA, CA 91016 U.S.A.
CATALOG NO. SEE LABEL ON REAR
MODEL NO. PANEL FOR MODEL AND
SERIAL NO. SERIAL INFORMATION
INPUT POWER: SINGLE PHASE AC
VOLTS: 90~130 180~260 Hz: 50/60
MAXIMUM RMS AMPS: 10
LASER OUTPUT: Yb FIBER AIMING:
MAXIMUM: 55 W <5 mW
OSCILLATION: PULSED/CW CW
WAVE LENGTH: 1060-1150 nm 635 nm
CENTRAL EMISSION WAVELENGTH: 1065 nm
PULSE OUTPUT FREQUENCY 2-500 kHz
COMPLIES WITH 21 CFR 1040.10
CLASS IV LASER PRODUCT

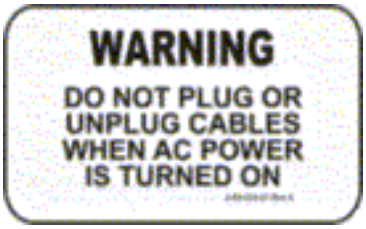
2


DANGER
HIGH VOLTAGE
DISCONNECT POWER AND WAIT 5 MIN.
BEFORE SERVICING.

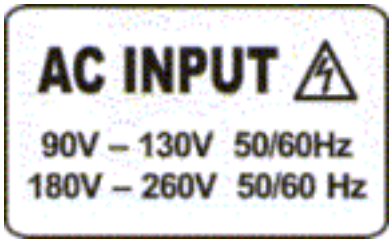
3


AMADA
AMADA MIYACHI AMERICA
MONROVIA, CA.
MADE IN U.S.A.
WWW.AMADAMIYACHI.COM
1-626-303-5676
CATALOG NO. LMF20
MODEL NO. 8-79-XXX-XXX
SERIAL NO. SERIAL NUMBER
INPUT POWER: SINGLE PHASE AC
VOLTS: 90~130 180~260 Hz: 50/60
MAXIMUM RMS AMPS: 10
CE
SEE LABEL ON SIDE PANEL FOR LASER SAFETY
AND COMPLIANCE DATA
4-66111-01 (REV A)

4

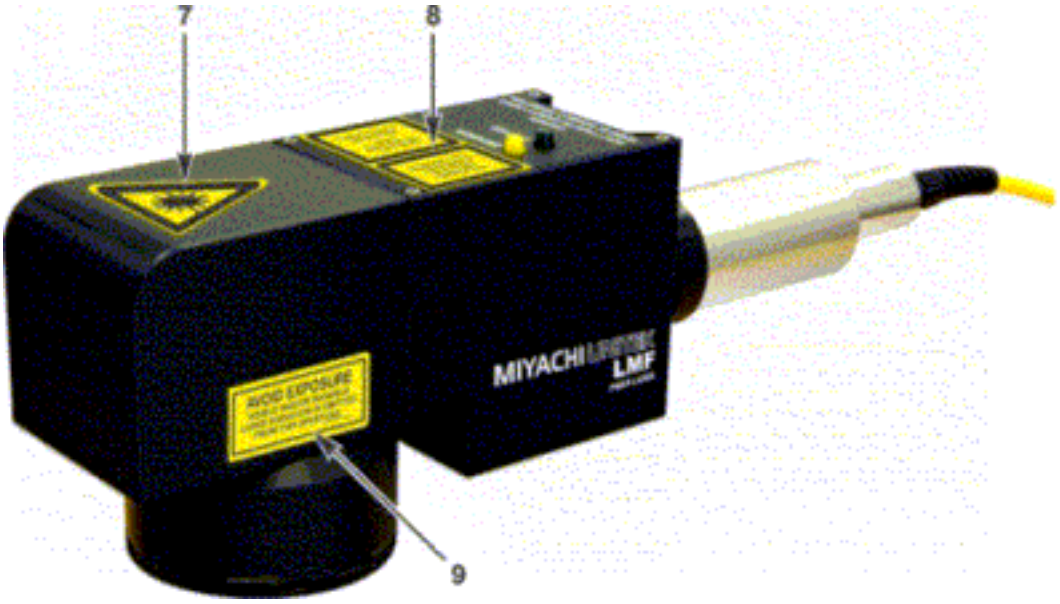

WARNING
DO NOT PLUG OR
UNPLUG CABLES
WHEN AC POWER
IS TURNED ON

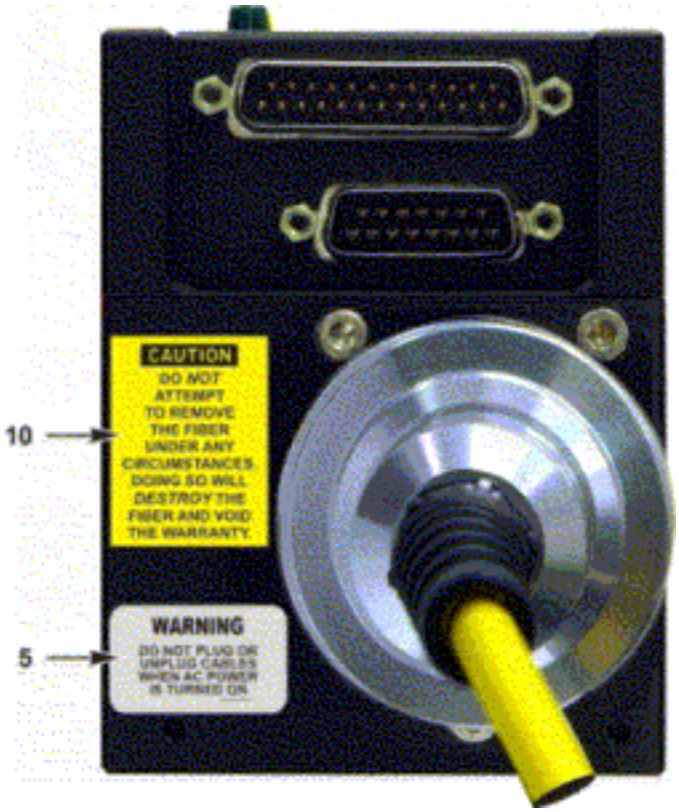
5


AC INPUT
90V – 130V 50/60Hz
180V – 260V 50/60 Hz

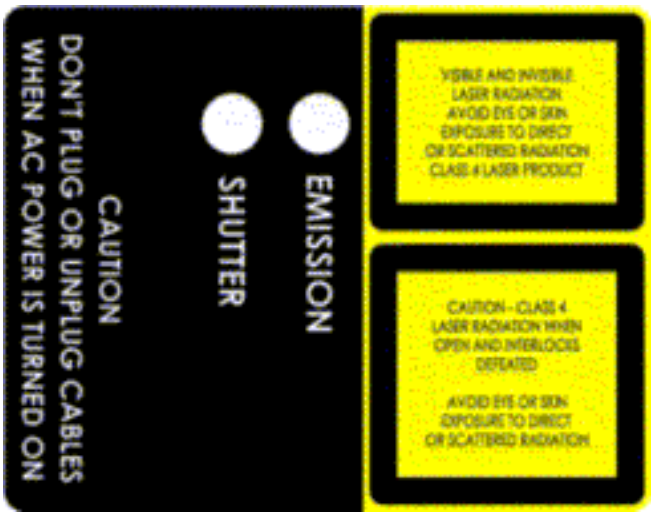
6

2D Head – Location on Ultra Compact Marker Head





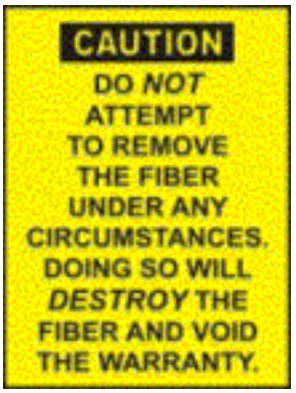
7


CAUTION
DON'T PLUG OR UNPLUG CABLES
WHEN AC POWER IS TURNED ON
SHUTTER
EMISSION
VISIBLE AND INVISIBLE
LASER RADIATION
AVOID EYE OR SKIN
EXPOSURE TO DIRECT
OR SCATTERED RADIATION
CLASS 4 LASER PRODUCT
CAUTION – CLASS 4
LASER RADIATION WHEN
OPEN AND INTERLOCKS
DEFEATED
AVOID EYE OR SKIN
EXPOSURE TO DIRECT
OR SCATTERED RADIATION

8

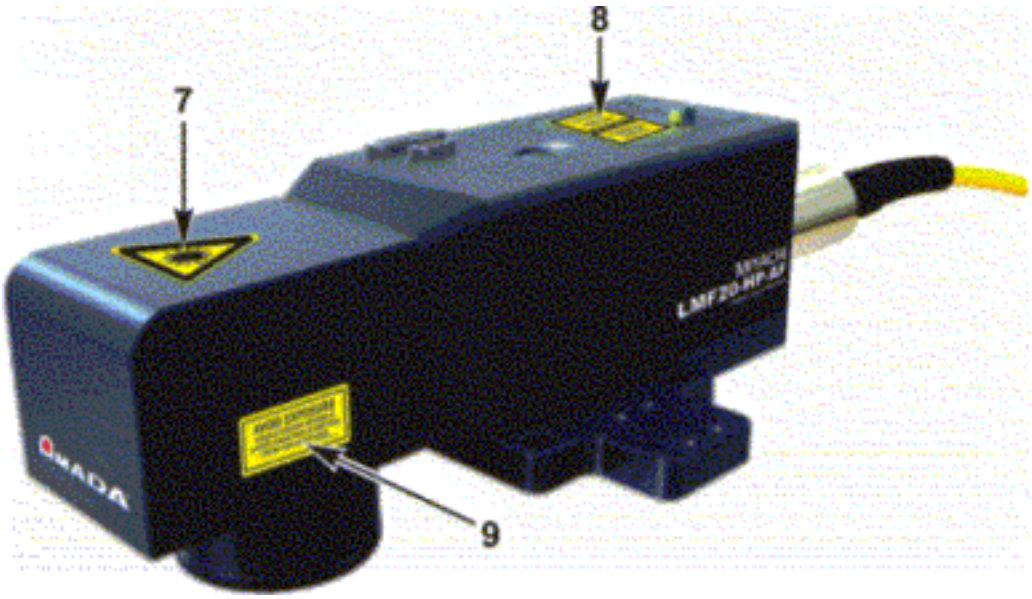

AVOID EXPOSURE
VISIBLE AND/OR INVISIBLE
LASER RADIATION IS EMITTED
FROM THIS APERTURE.

9

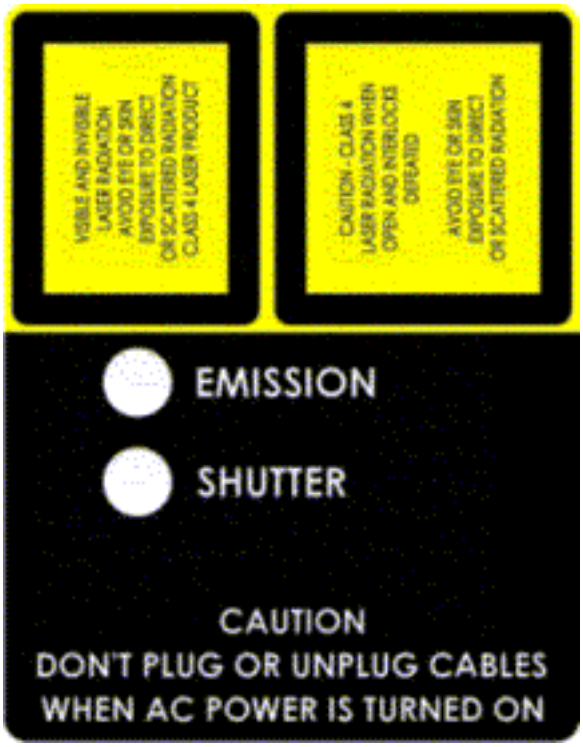

CAUTION
DO NOT
ATTEMPT
TO REMOVE
THE FIBER
UNDER ANY
CIRCUMSTANCES.
DOING SO WILL
DESTROY THE
FIBER AND VOID
THE WARRANTY.

10

AF Head – Location on AF (Adjustable Focus) Marker Head



7

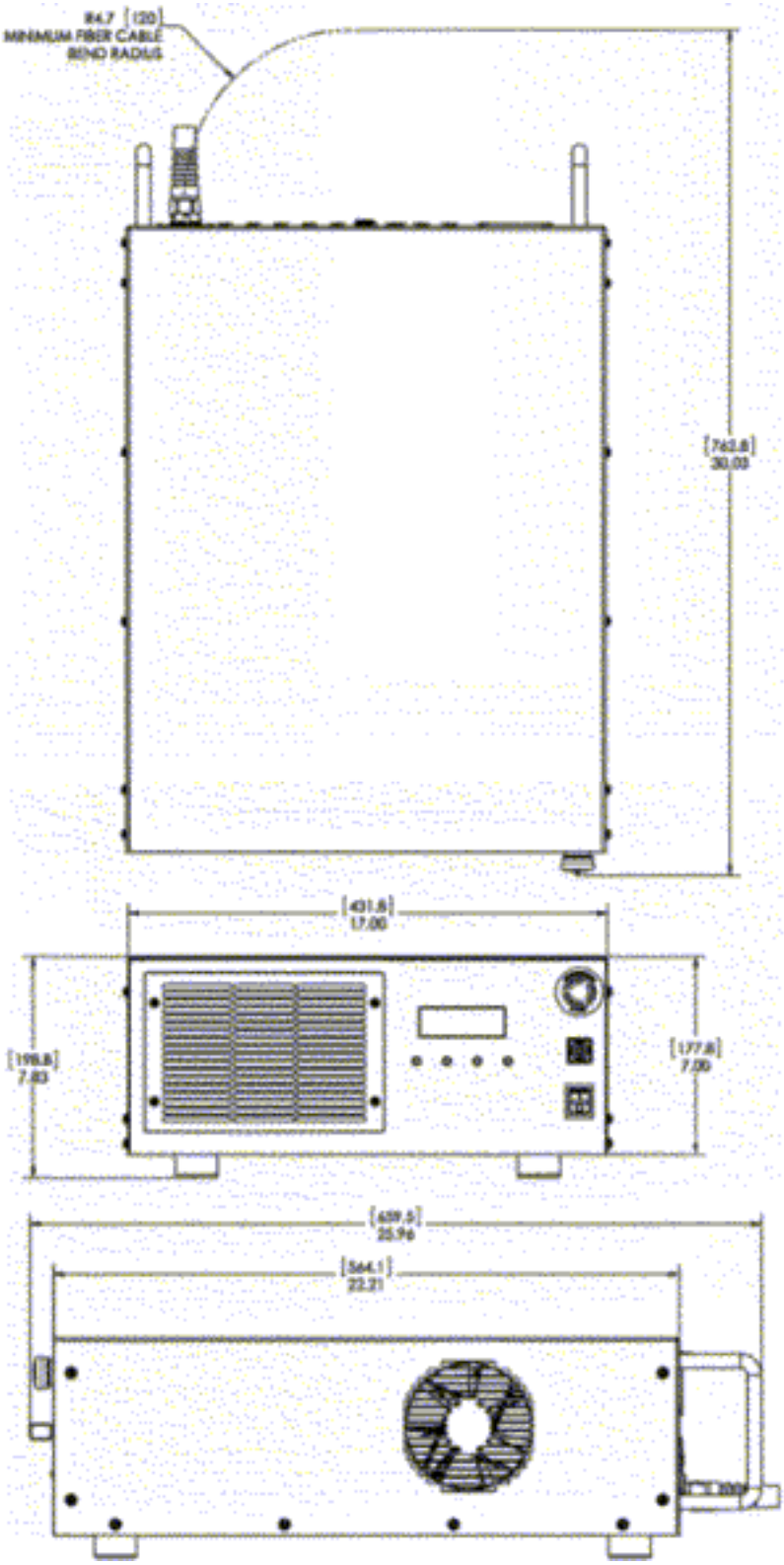


8

9

Section III. Engineering Drawings

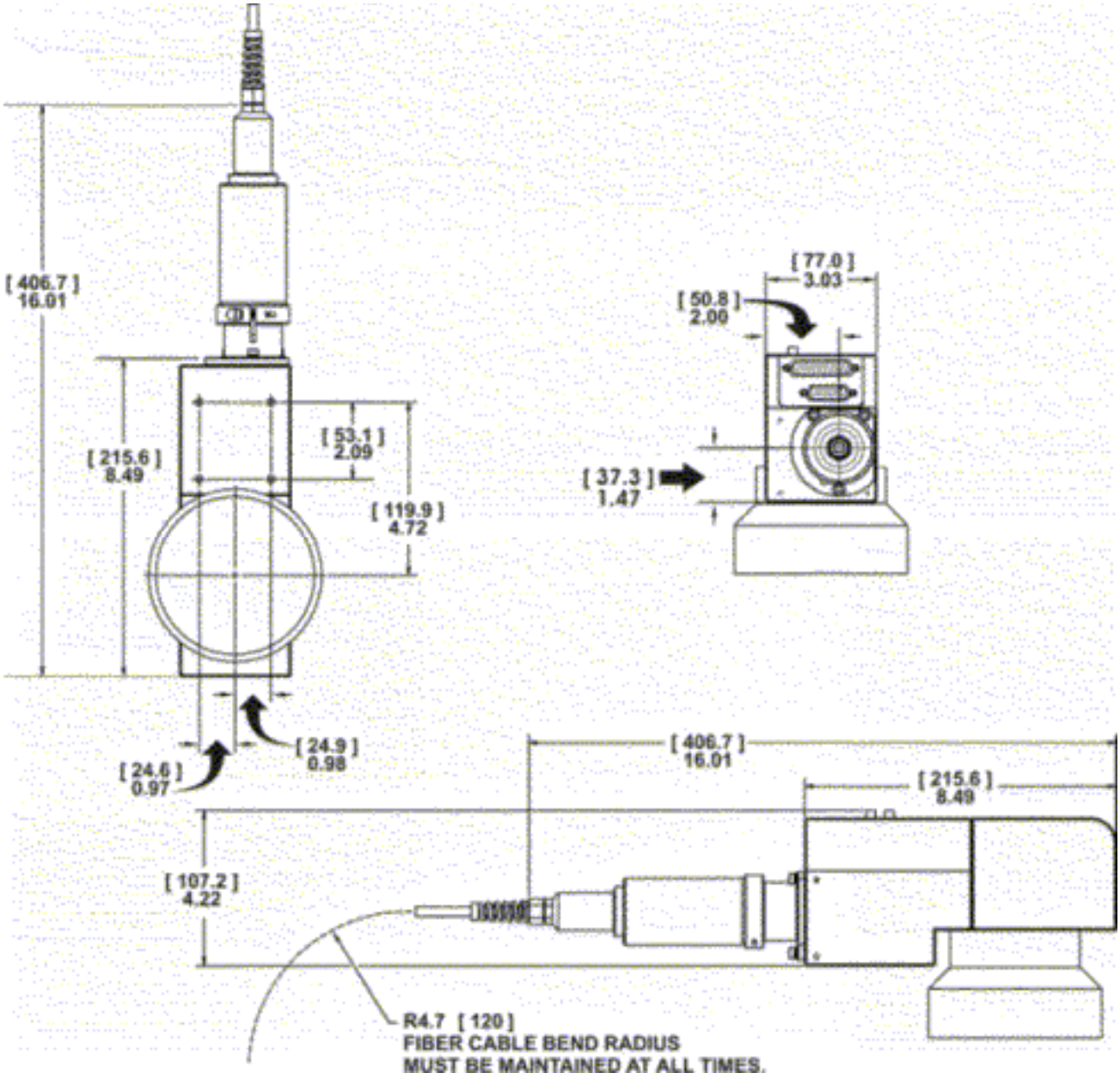
Control Unit



Installation Drawing for LMF50, LMF20, and LMF10

CAUTION

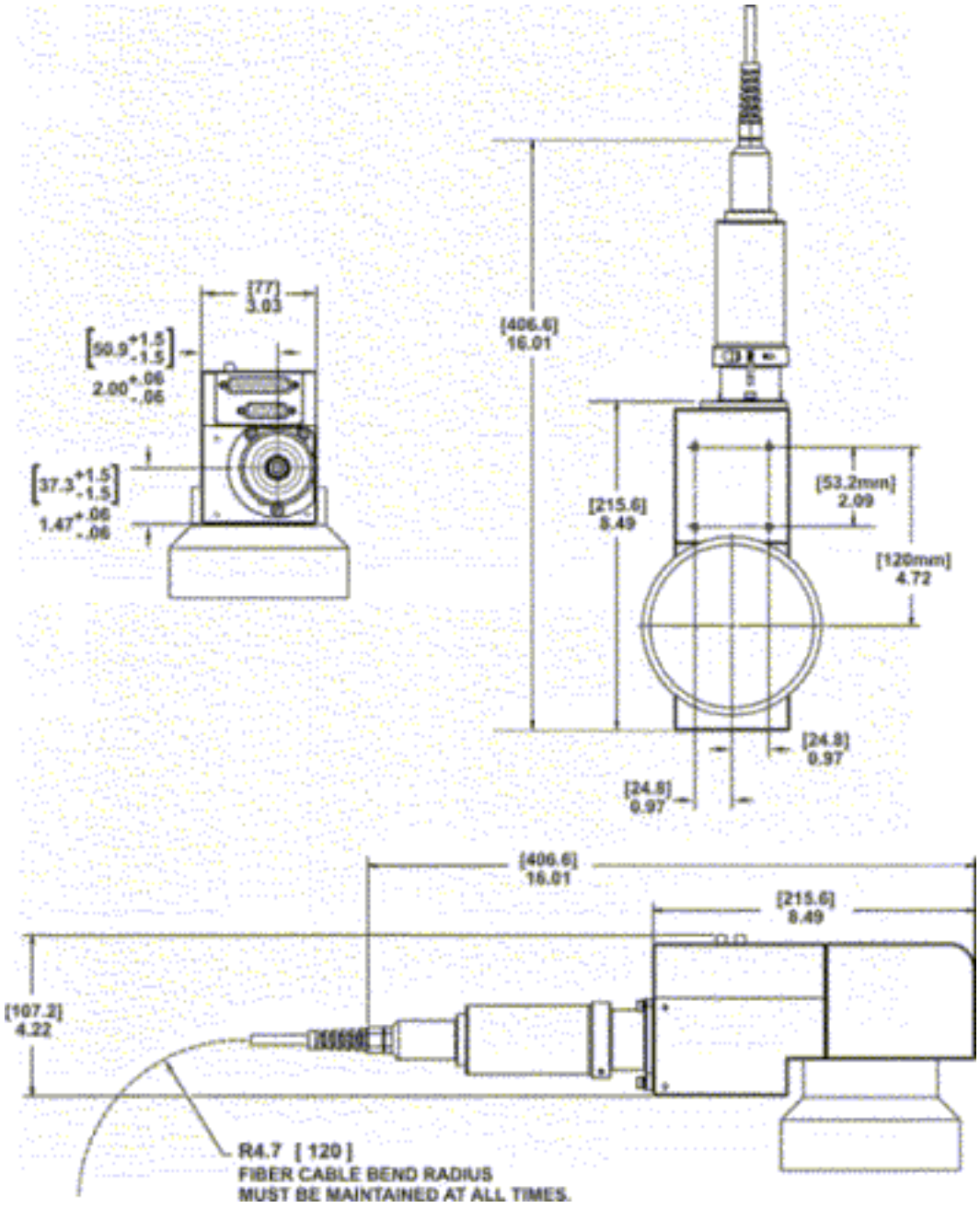
Do ***not*** attempt to remove the fiber at the rear of the marker head under any circumstances. Doing so will ***destroy*** the fiber and void the warranty. Amada Miyachi America assumes no liability for such action, the fiber will have to be replaced at the customer's expense.



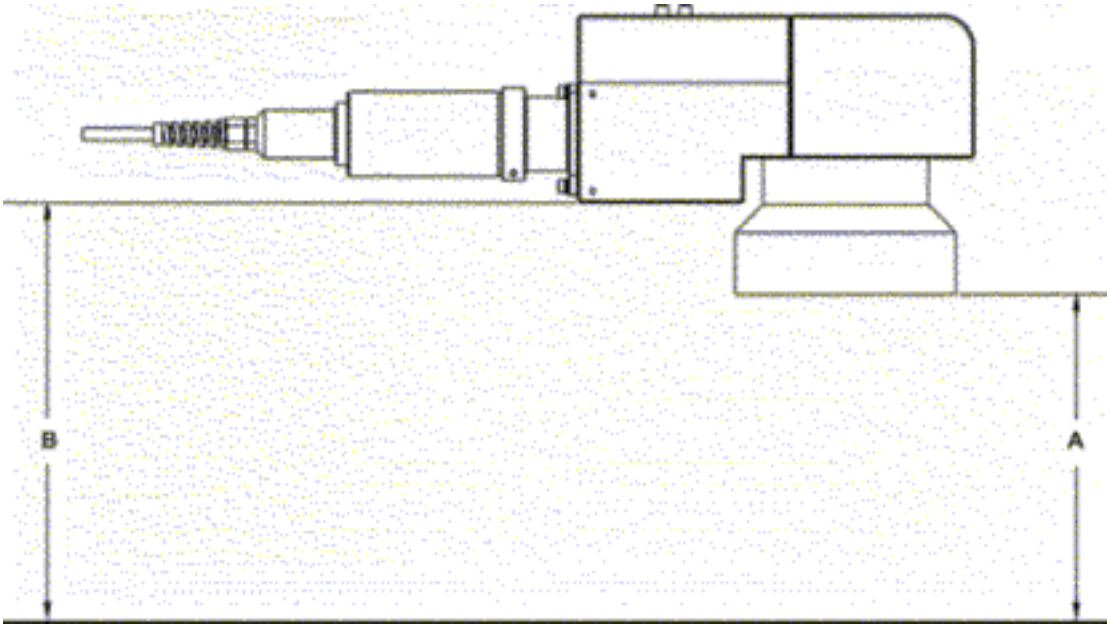
Installation Drawing for LMF20-SM, LMF20-HP, LMF35-HP and LMF70-HP

CAUTION

Do ***not*** attempt to remove the fiber at the rear of the marker head under any circumstances. Doing so will ***destroy*** the fiber and void the warranty. Amada Miyachi America assumes no liability for such action, the fiber will have to be replaced at the customer's expense.



2D Head - Ultra Compact High Performance Scanner Working Distance All Models

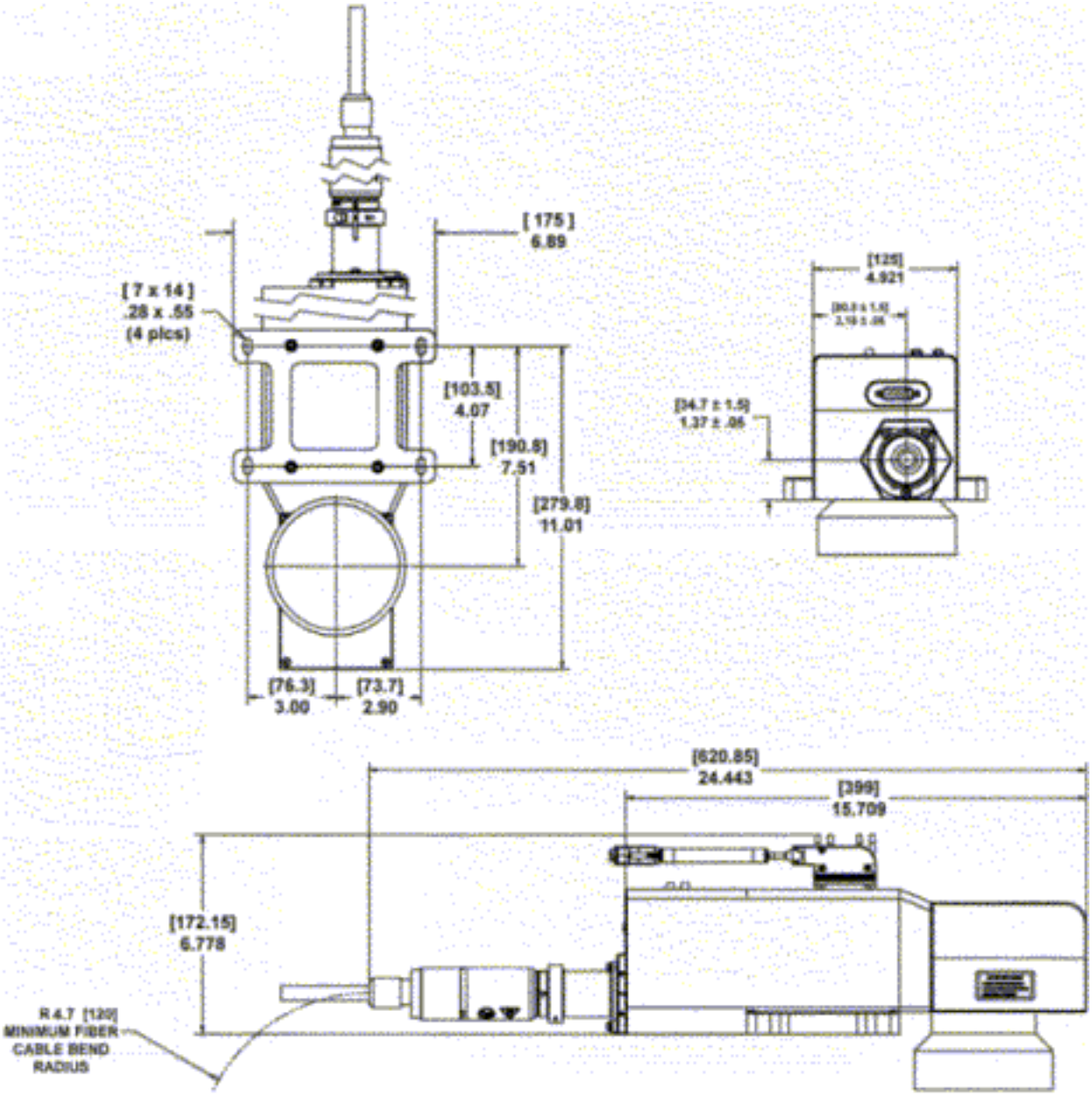


f- Theta Lens				
Dimension	**100mm**	**160mm**	**254mm**	**420mm**
A	3.86 ±.04” (98mm ±1mm)	6.93 ±.08” (176 ±2mm)	11.65 ±.12” (296 ±3mm)	19.45 ±.20” (494 ±5mm)
B	4.65 ±.04” (118.2 ±1mm)	7.72 ±.08” (196.2 ±2mm)	13.61 ±.12” (345.6 ±3mm)	21.63 ±.20” (549.3 ±5mm)

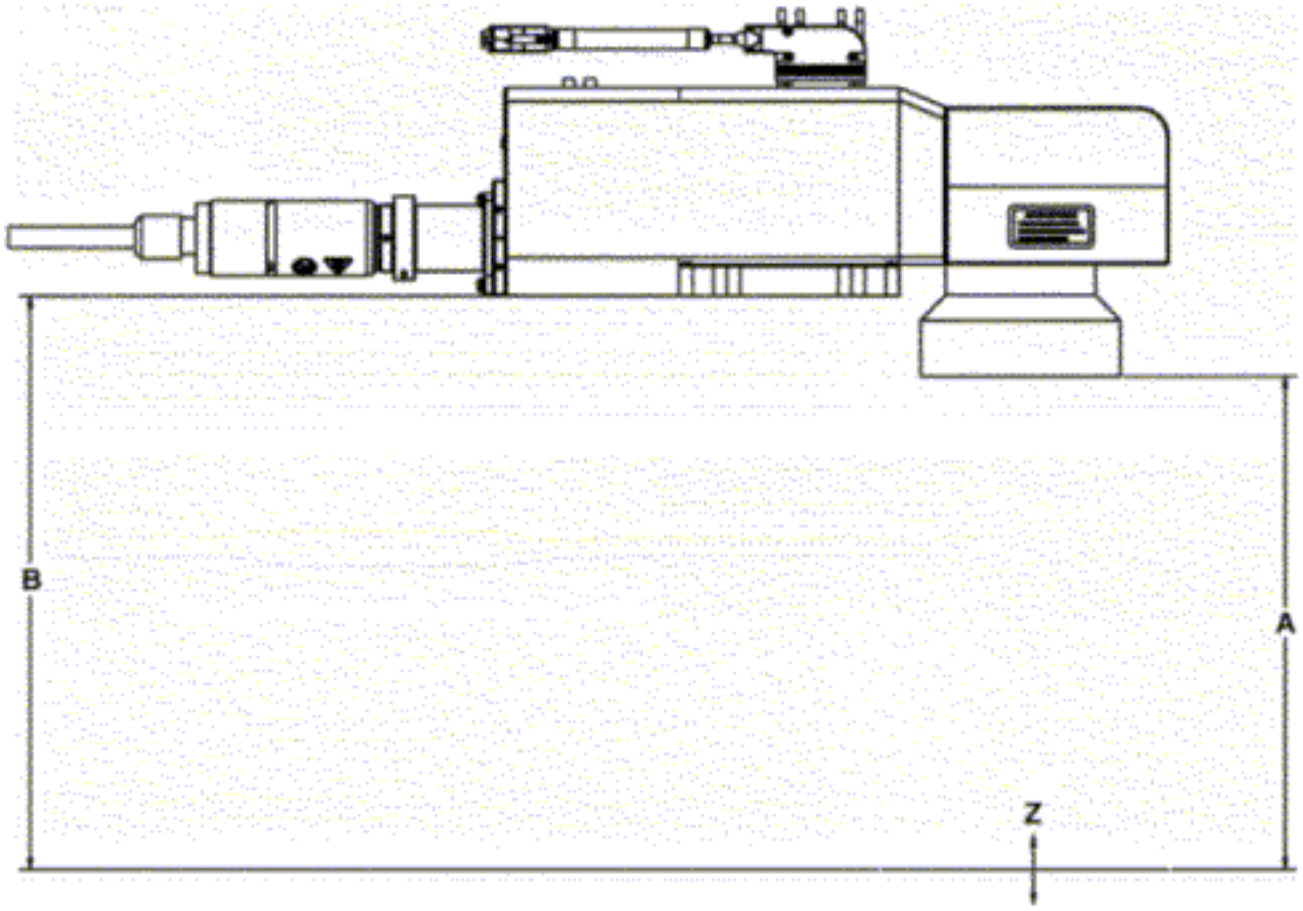
Installation Drawing for LMF with AF (Adjustable Focus) Head

CAUTION

Do ***not*** attempt to remove the fiber at the rear of the marker head under any circumstances. Doing so will ***destroy*** the fiber and void the warranty. Amada Miyachi America assumes no liability for such action, the fiber will have to be replaced at the customer's expense.



AF Head – AF (Adjustable Focus) Scanner Working Distance
All Models

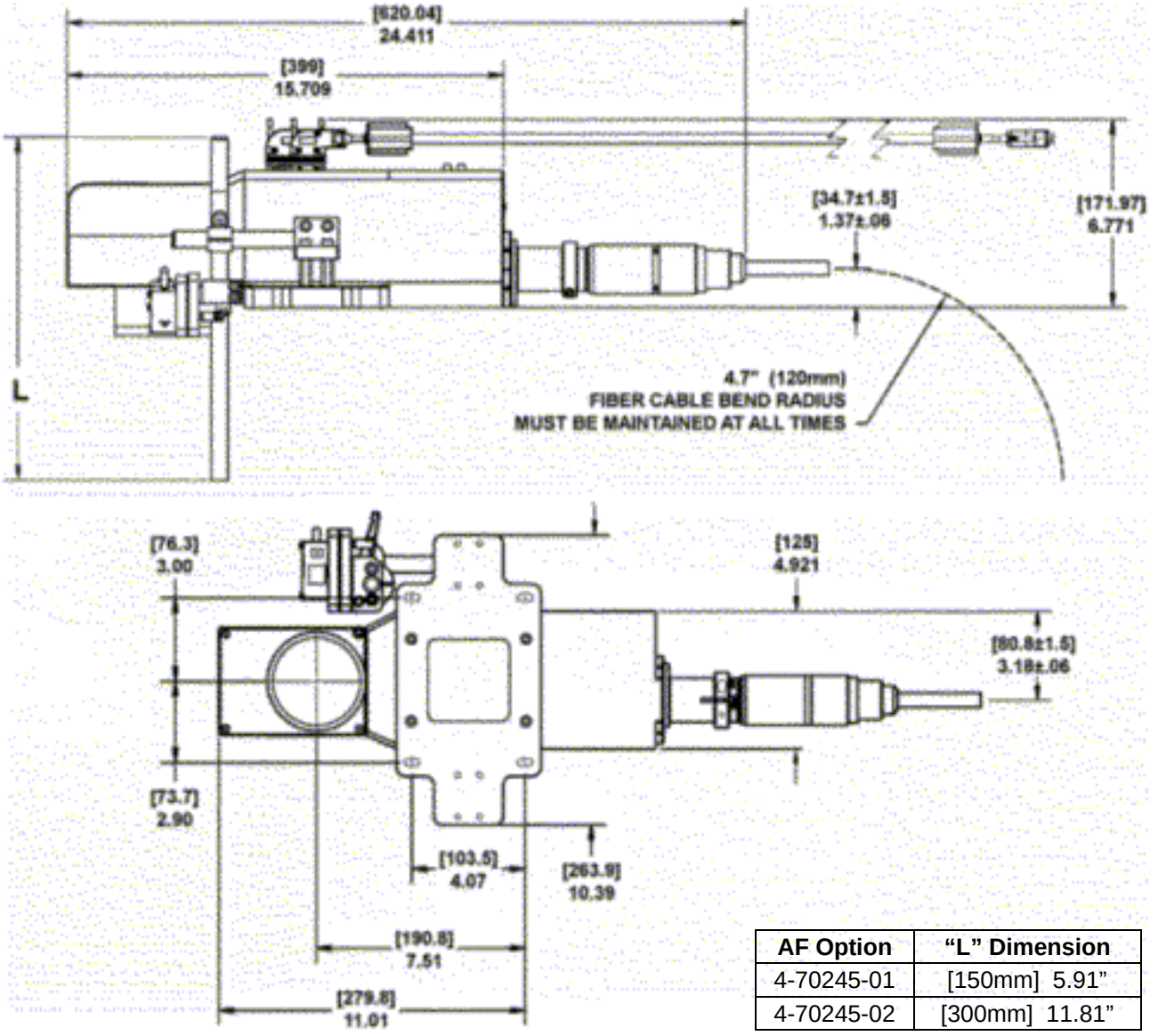


f- Theta Lens			
Dimension	**100mm**	**160mm**	**254mm**
A	3.84 ± .04” (97.5mm ± 1mm)	6.81 ± .08” (172.9 ± 2mm)	11.65 ± .12” (295.9 ± 3mm)
B	4.84 ± .04” (123.0 ±1mm)	7.81 ± .08” (198.5 ±2mm)	13.55 ± .12” (344.1 ±3mm)
Z	0.197” (± 5mm)	0.472” (± 12mm)	1.181” (± 30mm)

Installation Drawing for LMF with AF (Adjustable Focus) Head with Auto-Focus Bracket

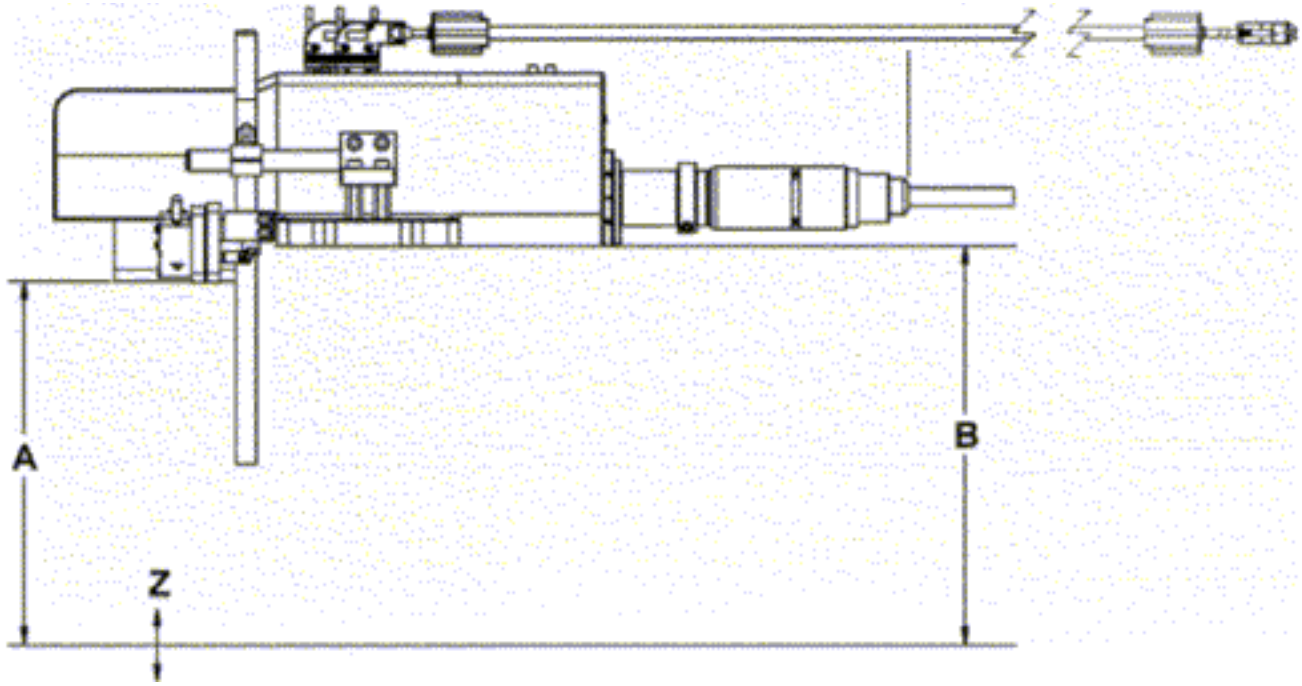
CAUTION

Do ***not*** attempt to remove the fiber at the rear of the marker head under any circumstances. Doing so will ***destroy*** the fiber and void the warranty. Amada Miyachi America assumes no liability for such action, the fiber will have to be replaced at the customer's expense.



AF Option	"L" Dimension
4-70245-01	[150mm] 5.91"
4-70245-02	[300mm] 11.81"

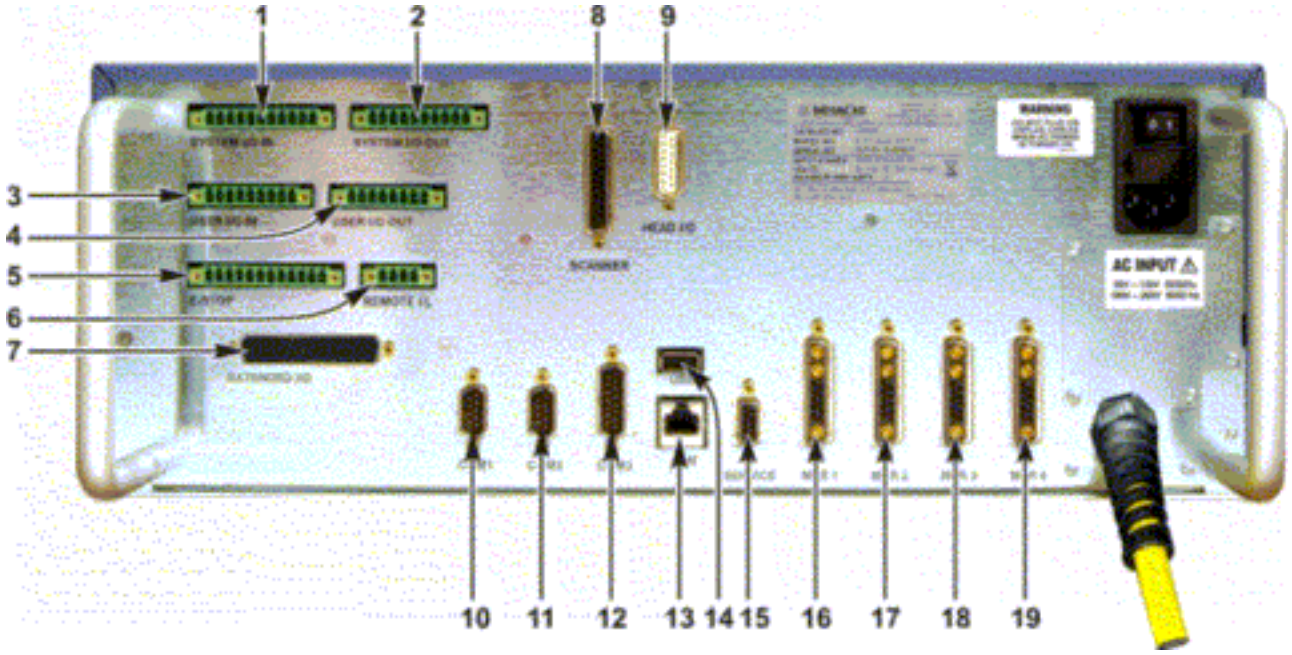
AF Head – AF (Adjustable Focus) Scanner with Auto-Focus Bracket Working Distance



f- Theta Lens			
Dimension	**100mm**	**160mm**	**254mm**
A	3.84 ± .04” (97.5mm ± 1mm)	6.81 ± .08” (172.9 ± 2mm)	11.65 ± .12” (295.9 ± 3mm)
B	4.84 ± .04” (123.0 ±1mm)	7.81 ± .08” (198.5 ±2mm)	13.55 ± .12” (344.1 ±3mm)
Z	0.197” (± 5mm)	0.472” (± 12mm)	1.181” (± 30mm)

APPENDIX B
ELECTRICAL AND DATA CONNECTIONS

Section I. Rear Panel Connectors



I/O Connections

This marker uses +24V logic as configured by the factory for I/O connections. By using the built-in I/O power supply, all inputs can be configured to accept either a sourced “high” (+24V) or a sinking “low” (0V) logic level and may be configured with switches, NPN or PNP transistors. Likewise all outputs can be configured to source a “high” (+24V) or sink a “low” (0V) logic level when turned ON. Both the System and User I/O Connectors include an opto-coupler bias input that is used to bias the I/O connector with source or sink capabilities. For added flexibility, an external Power Supply (+5 to +24V) may also be used to bias the I/O connections instead of using the internal +24V power supply. The maximum I/O output current is limited to 50mA (per output pin).

In all of the examples shown below, the internal +24V supply is used to bias all I/O bits to allow all inputs and outputs to “source”.

NOTE: Each +24V power supply pin must be wired to draw less than 500mA.

Phoenix Connectors

Connectors 1 through 4 on the Rear Panel are Phoenix-type connectors used for general purpose I/O. Depending on how these connectors are wired, the control can be configured several different ways in order to match your application needs. Configuration is achieved by using connectors with factory-installed jumpers, and by fabricating your own I/O cables.

Connectors 5 and 6 are Emergency Stop and Interlock connectors and should be configured appropriately using dry contact inputs.

Connector 7 is a 50 pin D-Subminiature connector that contains expanded user inputs and outputs. These are configured similar to the I/O Phoenix connectors, but must be interfaced with using a 50 pin D-Sub connector. I/O biasing guidelines apply to this connector as well.

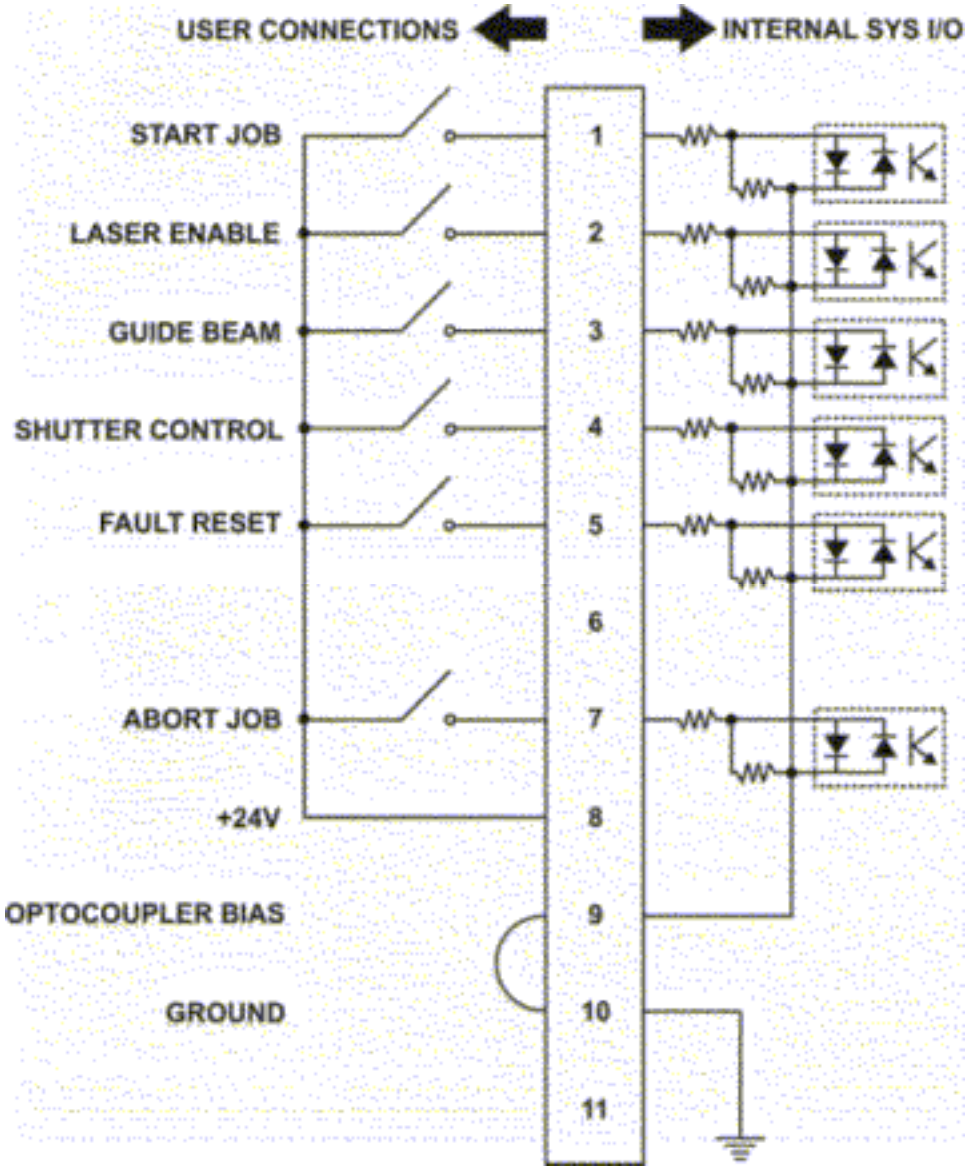
With the exception of connector 7 these connectors use screw-terminal wire connections. No soldering is required which makes configuring, or re-configuring, the connectors a simple task that can be done in just a few minutes. Connector 7 must be soldered or crimped depending on the choice of connector. Accessory connectors provided by Amada Miyachi America are solder-cup design.

Ensure that all connections are properly strain relieved to prevent pulling wires out of the connectors. Backshells are recommended on each connector.

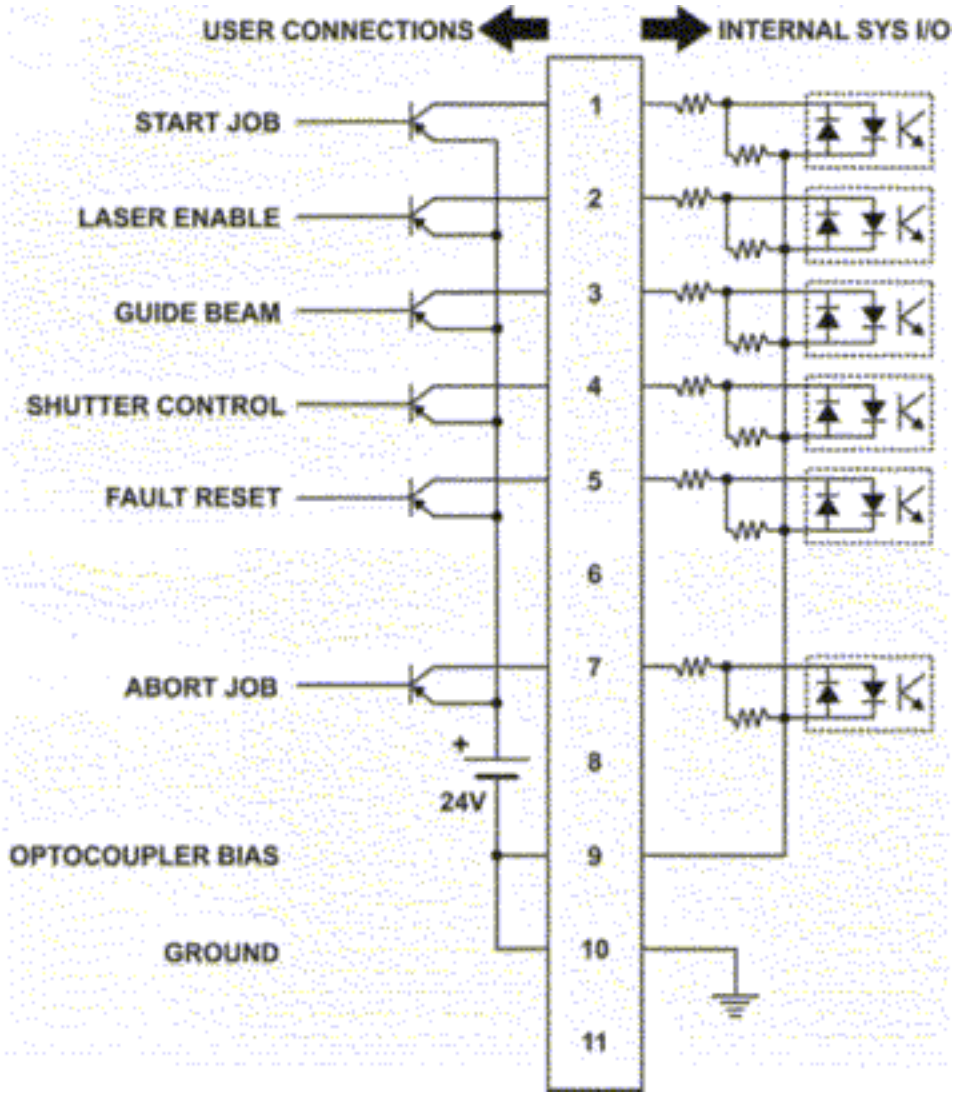
1. SYSTEM I/O IN Connector

Pin #	Signal	IN/OUT	Description	Note
1	Start Job	IN	**Start Job.** Begins the job sequence.	Photo coupler input
2	Laser Enable	IN	**Laser Enable.** Enables the fiber laser marker. No laser emission is possible without triggering this input. This input can be used as a high speed process interrupt for resistor trimming or other similar applications.	Photo coupler input
3	Guide Beam	IN	**Guide Beam Control.** Allows external control of visible guide beam NOTE: markers cannot emit laser radiation if the guide beam input is enabled. Turn off the guide beam before initiating process.	Photo coupler input
4	External Shutter Control	IN	**External Shutter Control.** Allows external control of the safety shutter. This input must be triggered for the shutter to open and permit laser emission.	Photo coupler input
5	Fault Reset	IN	**Fault Reset**. Resets any fault conditions.	Photo coupler input
6	Unused	—	*Unused*	—
7	Abort Job	IN	**Abort Job.** Aborts the current job.	Photo coupler input
8	+24 VDC	—	**+24V**	—
9	DI Common	—	**Optocoupler Bias Input**	—
10	0V	—	**GROUND**	—
11	Unused	—	*Unused*	—

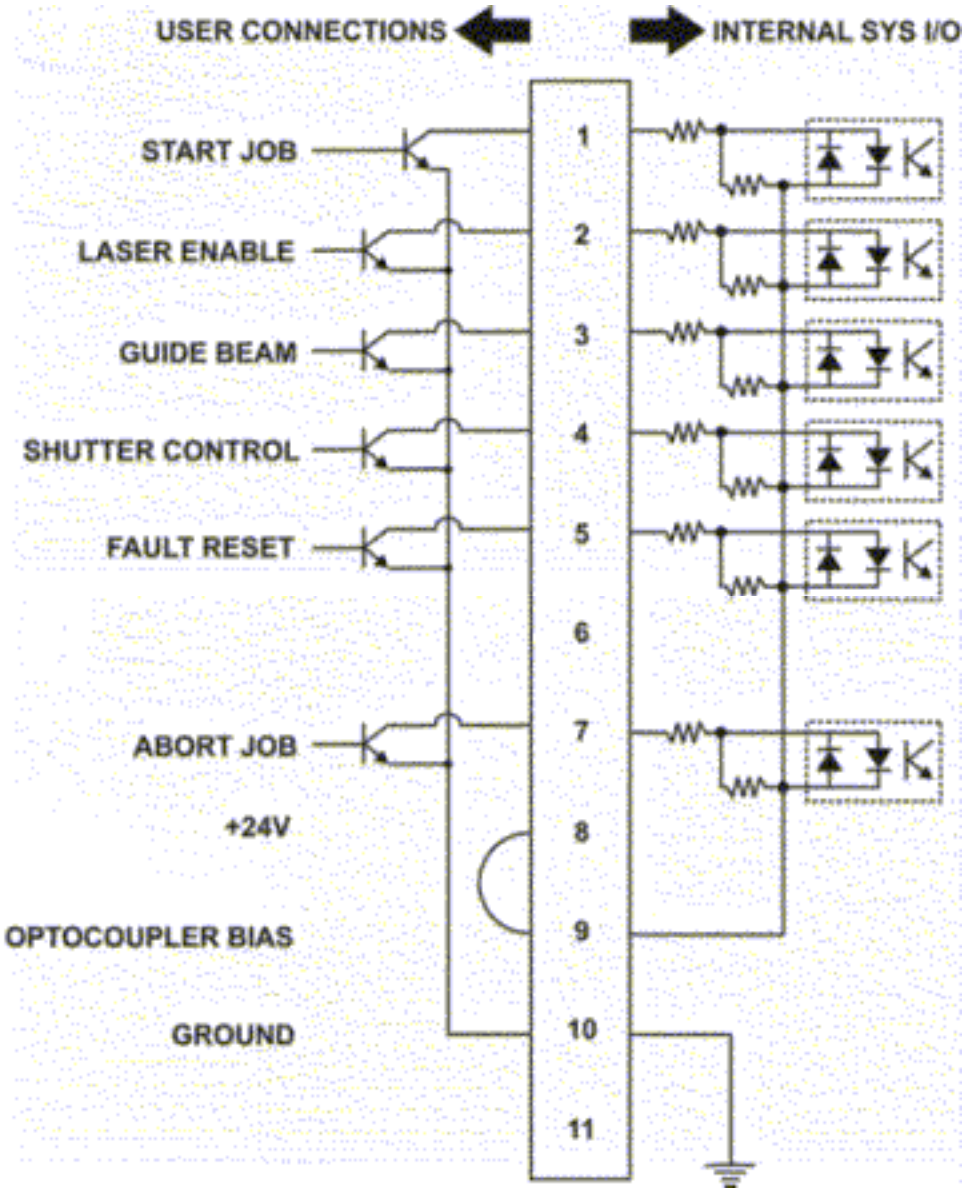
System I/O IN • Dry Contact Input Switches (internally biased)



System I/O IN • Transistor (PNP) Input with External +24 VDC Power Source



System I/O IN • Transistor (NPN) Input (internally biased)

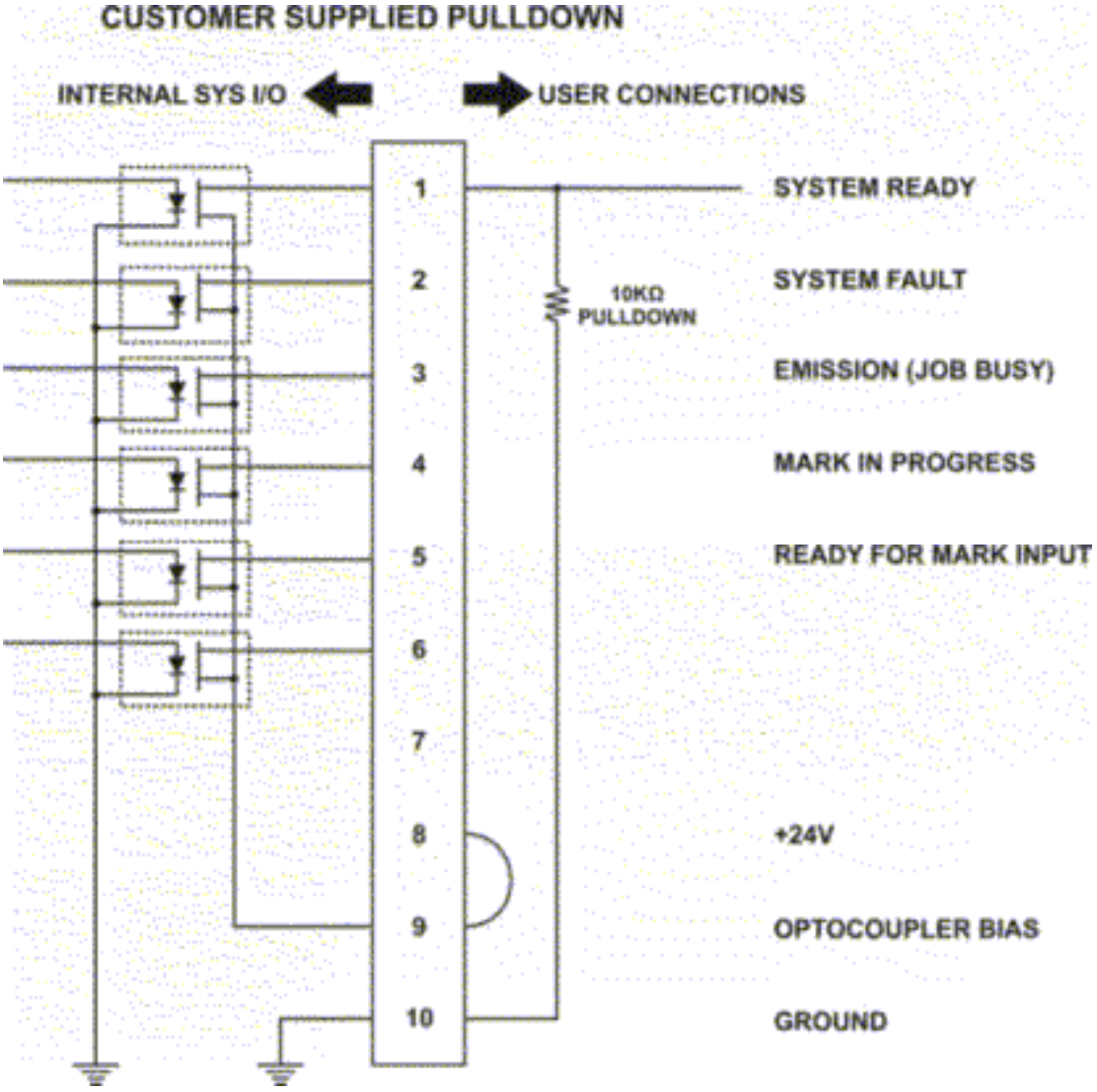


2. SYSTEM I/O OUT Connector

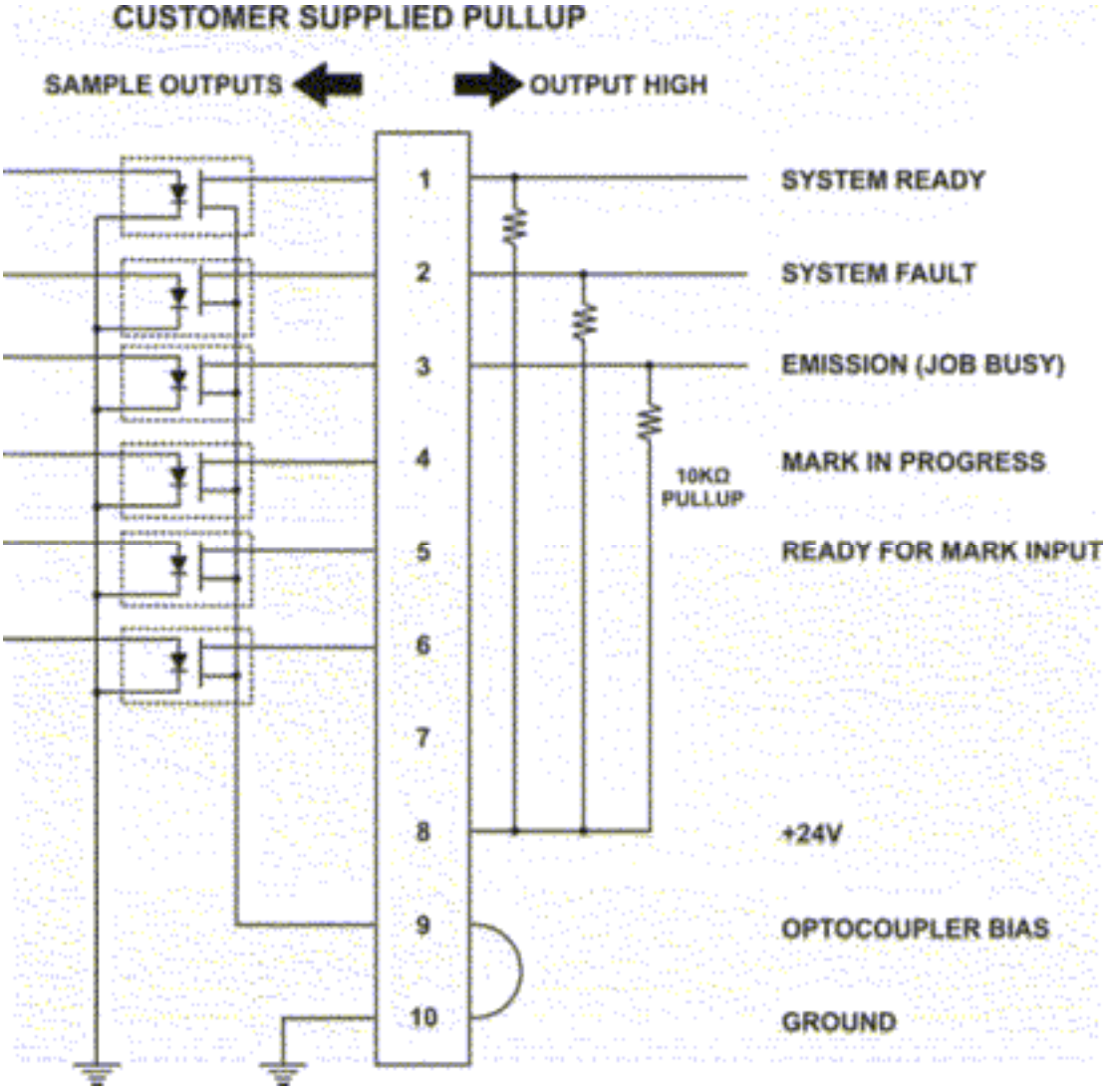
All outputs are configured with photo-MOS outputs. In order to properly use these outputs, they will need to be biased by an external resistor. In the examples below, a 10kΩ resistor is used to keep the output from floating. The Outputs can either sink or source depending on how the optocoupler bias jumper and resistors are configured.

Pin #	Signal	IN/OUT	Description	Note
1	System Ready	OUT	**System Ready.** The system is ready for laser processing and the laser control hardware has booted and obtained an IP address. No system faults are present.	Photo MOS output
2	System Fault	OUT	**System Fault.** A fault condition exists that must be cleared.	Photo MOS output
3	Job Busy	OUT	**Laser Emission In Progress (Job Busy).** A laser marking job is currently being executed. This output will be active from the time the job is executed until it is completed.	Photo MOS output
4	Mark in Progress	OUT	**Mark In Progress**. The laser is currently marking a segment. This output will trigger on and off rapidly as the laser marks the various components of a job.	Photo MOS output
5	Ready for Mark	OUT	**Ready for Mark Input.** A job has been loaded and is ready to execute. This output is intended for use	Photo MOS output
6	Unused	—	*Unused*	—
7	Unused	—	*Unused*	—
8	+24 VDC Output	—	**+24V**	—
9	DO Common	—	**Optocoupler Bias Input**	—
10	0V	—	**GROUND**	—

System I/O OUT • Pulldown Output



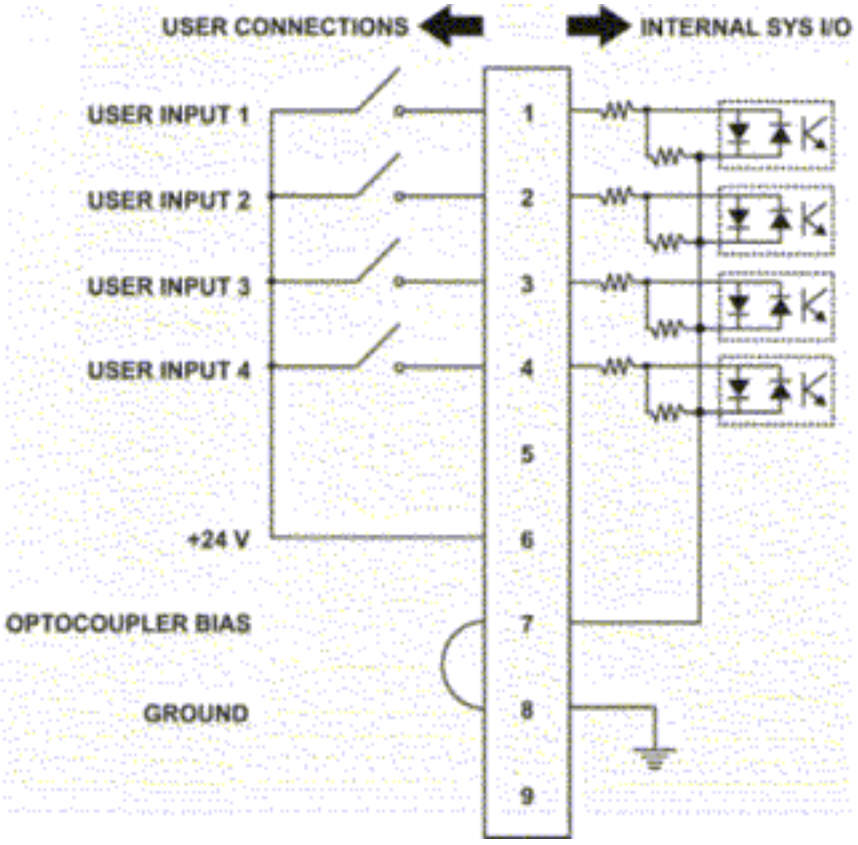
System I/O OUT • Pullup Output



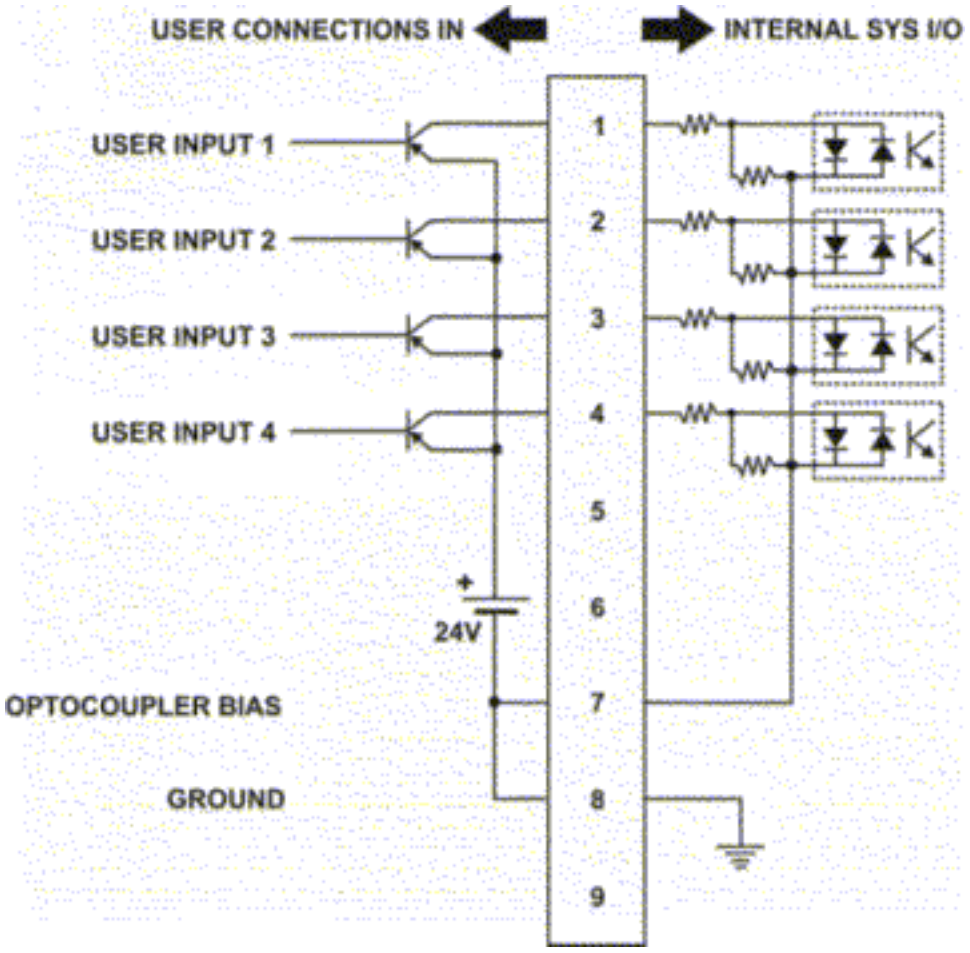
3. USER I/O IN Connector

Pin #	Signal	IN/OUT	Description	Note
1	User Input 1	IN	**User Input 1**	Photo coupler input
2	User Input 2	IN	**User Input 2**	Photo coupler input
3	User Input 3	IN	**User Input 3**	Photo coupler input
4	User Input 4	IN	**User Input 4**	Photo coupler input
5	Unused	—	*Unused*	—
6	+24 VDC	—	**+24V**	—
7	DI Common	—	**Optocoupler Bias Input**	—
8	0V	—	**GROUND**	—
9	Unused	—	*Unused*	—

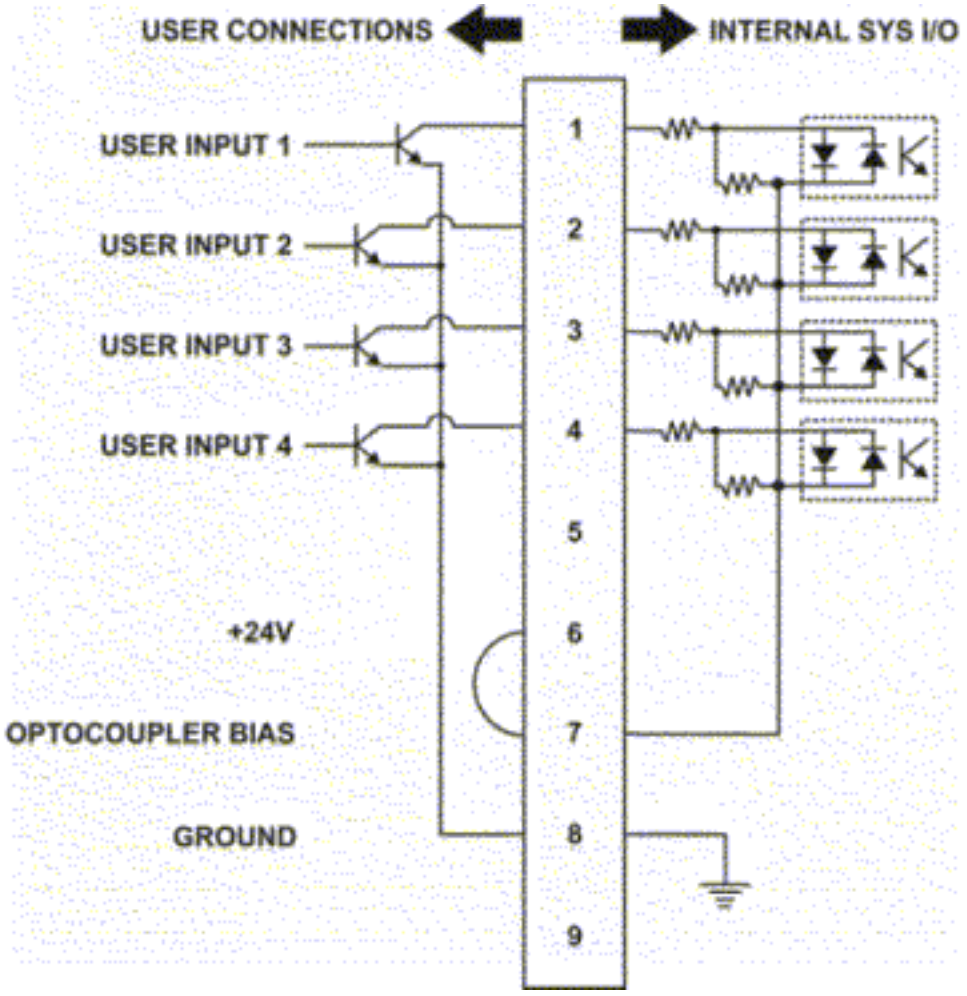
User I/O IN • Dry Contact Input Switches (internally biased)



User I/O IN • Transistor (PNP) Input with External +24 VDC Power Source



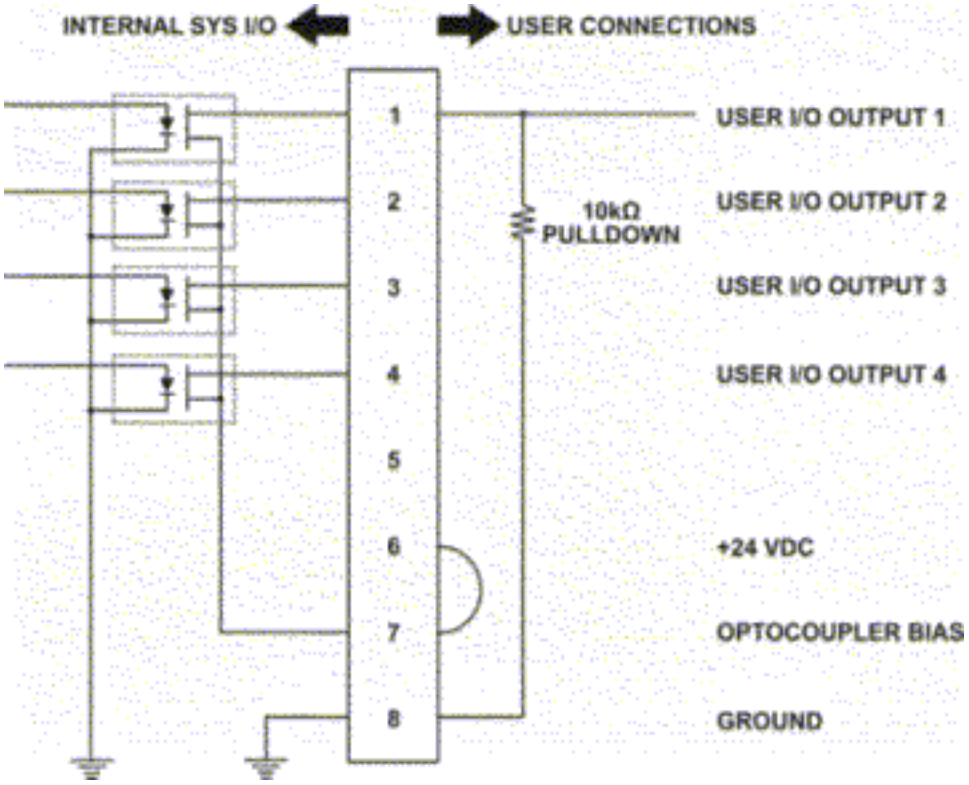
User I/O IN • Transistor (NPN) Input (internally biased)



4. USER I/O OUT Connector

Pin #	Signal	IN/OUT	Description	Note
1	User Output 1	OUT	**User Output 1**	Photo coupler input
2	User Output 2	OUT	**User Output 2**	Photo coupler input
3	User Output 3	OUT	**User Output 3**	Photo coupler input
4	User Output 4	OUT	**User Output 4**	Photo coupler input
5	*Unused*	—	*Unused*	—
6	+24 VDC Output	—	**+24V**	—
7	DO Common	—	**Optocoupler Bias Input**	—
8	0V	—	**GROUND**	—

User I/O OUT • Pulldown Output



WARNING

Proper integration of the fiber laser marker and external equipment is required for compliance with applicable safety regulations. Failure to select and implement a correct method of wiring can render the fiber laser marker unsafe.

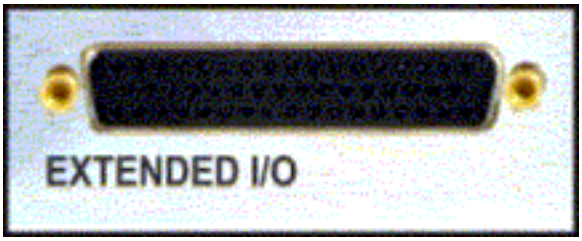
5. E-STOP Connector

See *Chapter 2, Section III* for information on integrating the Emergency Stop circuit into the overall machine emergency stop circuit.

6. Remote Interlock Connector

See *Chapter 2, Section III* for information on integrating the Remote Interlock circuit into the overall machine emergency stop circuit.

7. Extended I/O and Job Select Connector

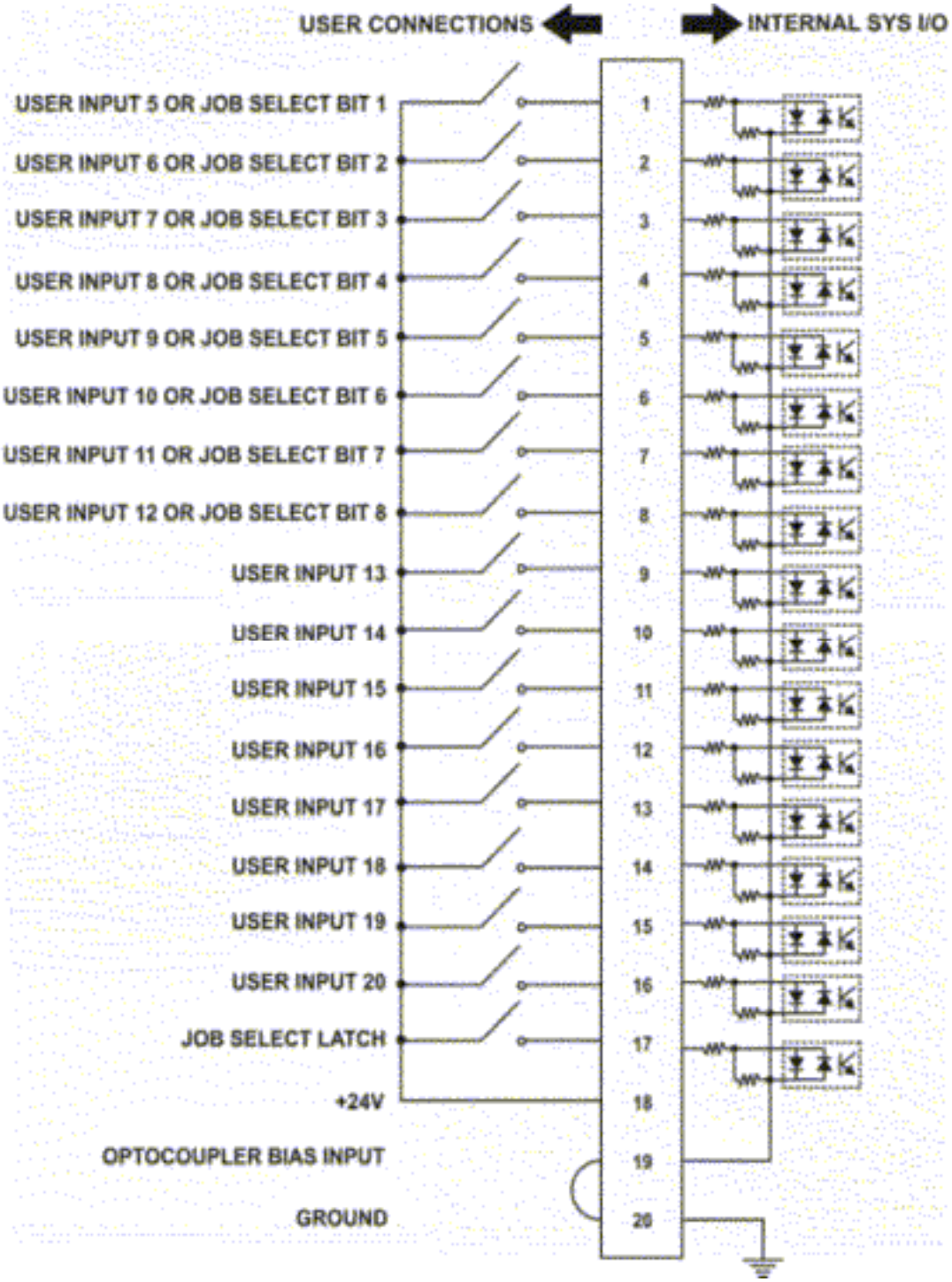


NOTE: In Job Select Mode inputs 5-12 are used to load laser jobs from laser internal or USB storage. Otherwise they function as user process inputs.

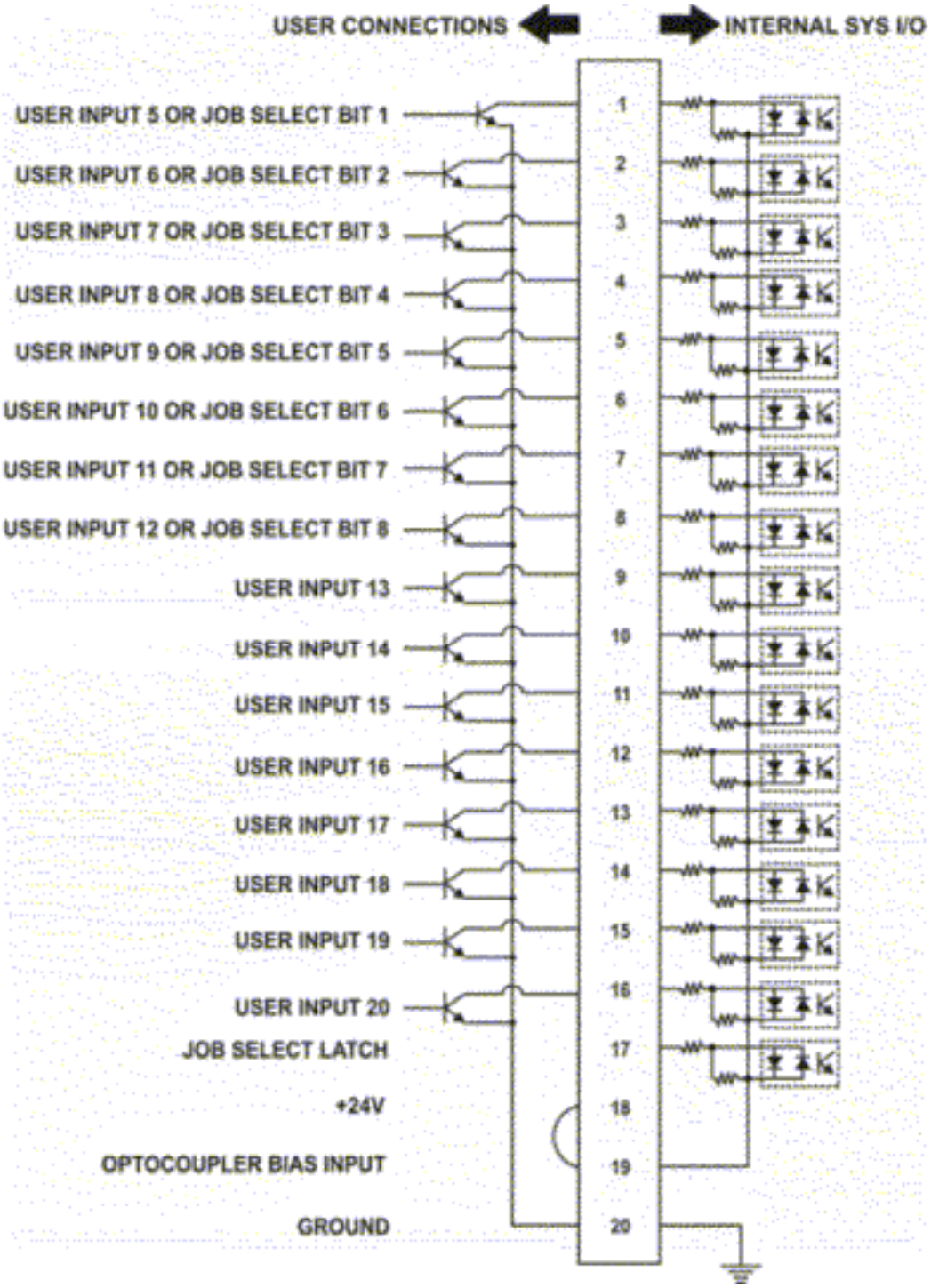
Extended I/O and Job Select Connector – Input Pins

Pin #	Signal	IN/OUT	Description	Note
1	User Input 5 or Job Select Bit 1	IN	**User Input 5 or Job Select Bit 1**	Photo coupler input
2	User Input 6 or Job Select Bit 2	IN	**User Input 6 or Job Select Bit 2**	Photo coupler input
3	User Input 7 or Job Select Bit 3	IN	**User Input 7 or Job Select Bit 3**	Photo coupler input
4	User Input 8 or Job Select Bit 4	IN	**User Input 8 or Job Select Bit 4**	Photo coupler input
5	User Input 9 or Job Select Bit 5	IN	**User Input 9 or Job Select Bit 5**	Photo coupler input
6	User Input 10 or Job Select Bit 6	IN	**User Input 10 or Job Select Bit 6**	Photo coupler input
7	User Input 11 or Job Select Bit 7	IN	**User Input 11 or Job Select Bit 7**	Photo coupler input
8	User Input 12 or Job Select Bit 8	IN	**User Input 12 or Job Select Bit 8**	Photo coupler input
9	User Input 13	IN	**User Input 13**	Photo coupler input
10	User Input 14	IN	**User Input 14**	Photo coupler input
11	User Input 15	IN	**User Input 15**	Photo coupler input
12	User Input 16	IN	**User Input 16**	Photo coupler input
13	User Input 17	IN	**User Input 17**	Photo coupler input
14	User Input 18	IN	**User Input 18**	Photo coupler input
15	User Input 19	IN	**User Input 19**	Photo coupler input
16	User Input 20	IN	**User Input 20**	Photo coupler input
17	Job Select Bit Latch	IN	**Job Select Latch**	Used to latch bit combinations for job loads; Photo coupler input
18	+24V DC Output	—	**+24V**	
19	EXT DI COMMON	IN	**Optocoupler Bias Input**	
20	0V	—	**GROUND**	

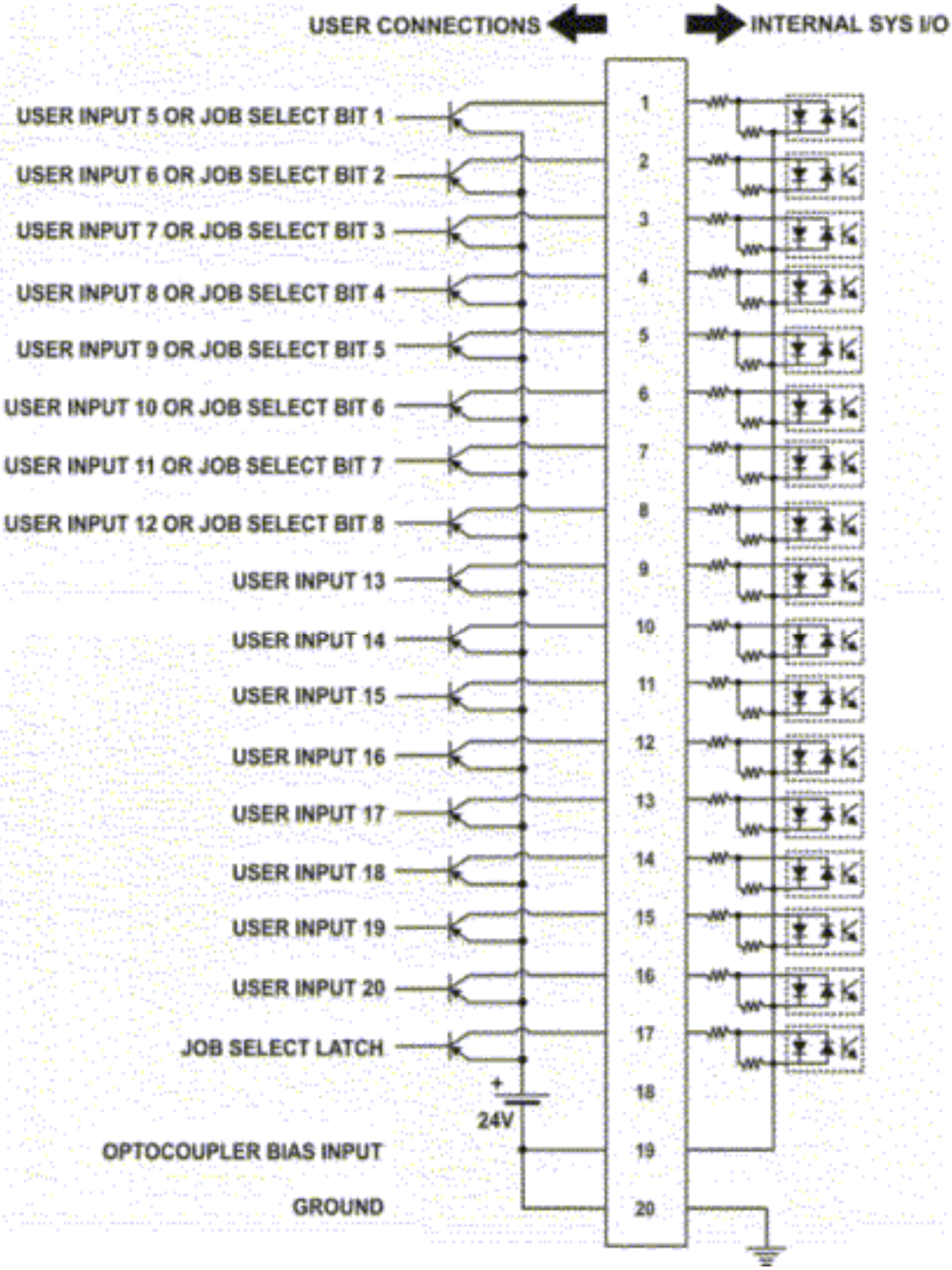
Extended I/O / Job Select • Dry Contact Input Switches (internally biased)



Extended I/O / Job Select • Transistor (NPN) Input (internally biased)



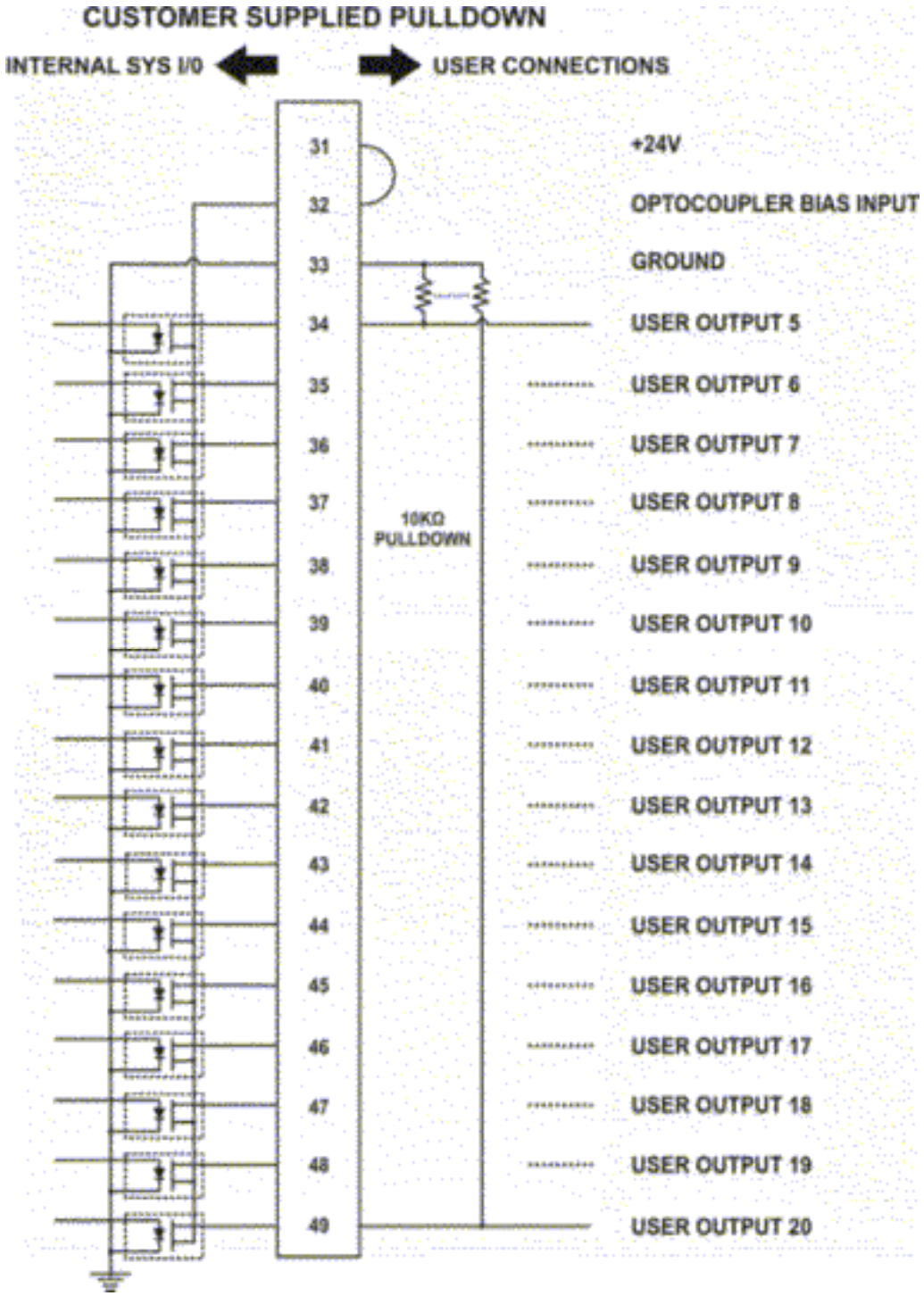
Extended I/O / Job Select • Transistor (PNP) Input with External +24V Power Source



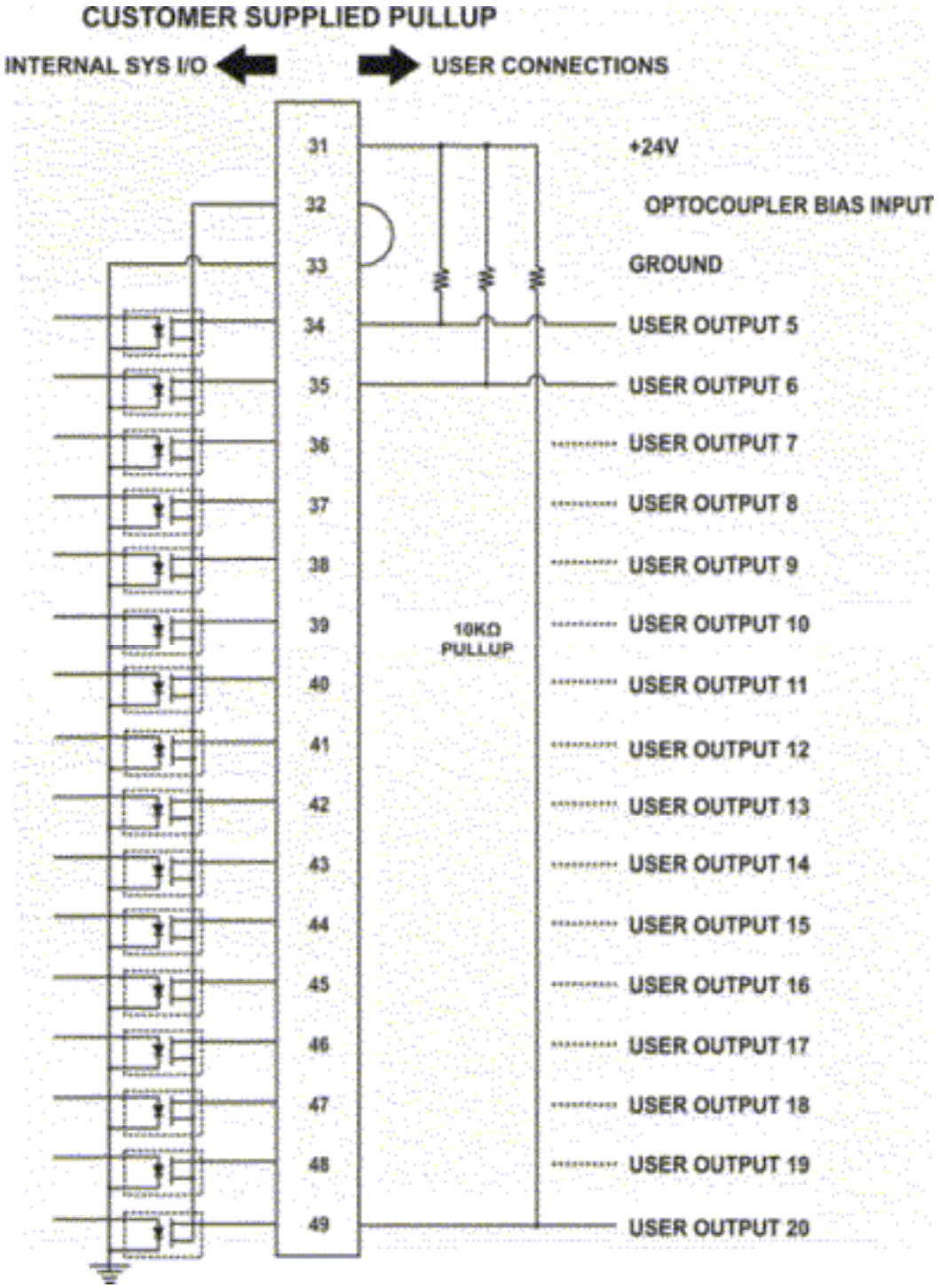
Extended I/O and Job Select Connector – Output Pins

Pin #	Signal	IN/OUT	Description	Note
31	+24V DC Output	—	**+24V**	
32	EXT DO COMMON	OUT	**Optocoupler Bias Input**	
33	0V	—	**GROUND**	
34	User Output 5	OUT	**User Output 5**	Photo MOS Output
35	User Output 6	OUT	**User Output 6**	Photo MOS Output
36	User Output 7	OUT	**User Output 7**	Photo MOS Output
37	User Output 8	OUT	**User Output 8**	Photo MOS Output
38	User Output 9	OUT	**User Output 9**	Photo MOS Output
39	User Output 10	OUT	**User Output 10**	Photo MOS Output
40	User Output 11	OUT	**User Output 11**	Photo MOS Output
41	User Output 12	OUT	**User Output 12**	Photo MOS Output
42	User Output 13	OUT	**User Output 13**	Photo MOS Output
43	User Output 14	OUT	**User Output 14**	Photo MOS Output
44	User Output 15	OUT	**User Output 15**	Photo MOS Output
45	User Output 16	OUT	**User Output 16**	Photo MOS Output
46	User Output 17	OUT	**User Output 17**	Photo MOS Output
47	User Output 18	OUT	**User Output 18**	Photo MOS Output
48	User Output 19	OUT	**User Output 19**	Photo MOS Output
49	User Output 20	OUT	**User Output 20**	Photo MOS Output

System I/O OUT • Pulldown Output



System I/O OUT • Pullup Output



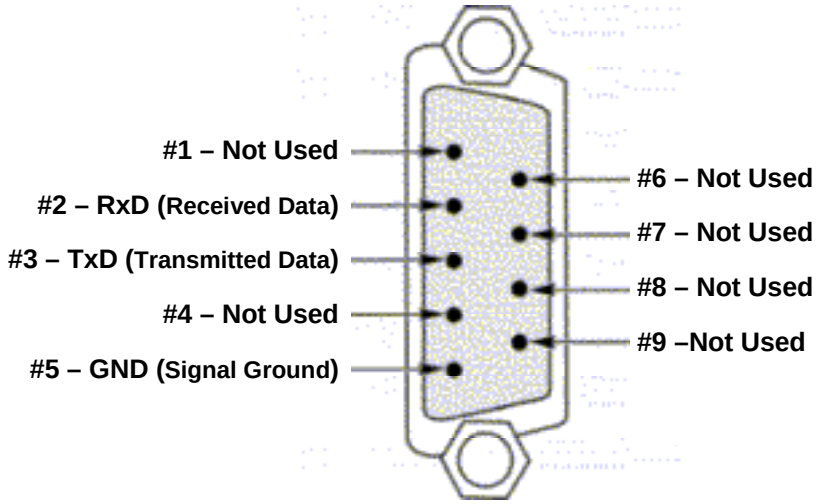
8. SCANNER Connector

Connects to the Laser Head.

9. LASER HEAD I/O Connector

Connects to the Laser Head.

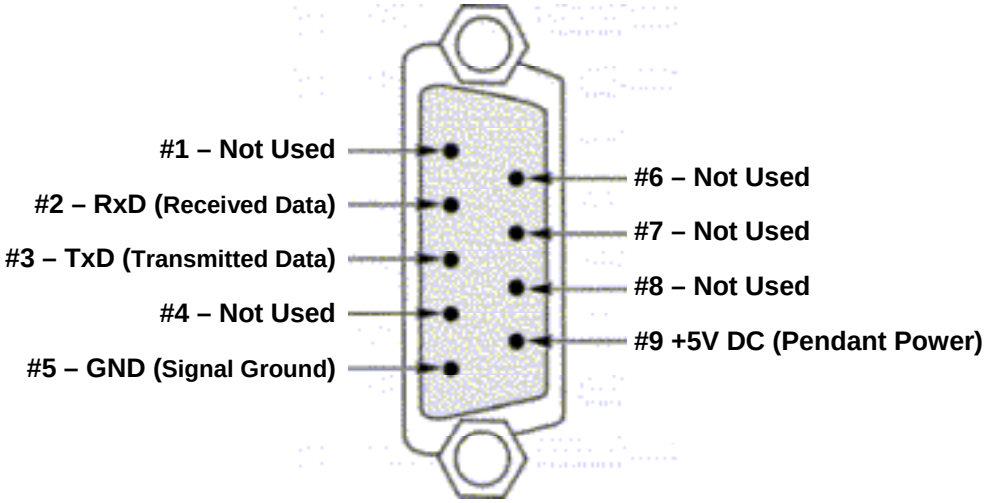
10. COM1 Connector



NOTES:

- The unit communicates via a simplified 3-wire RS-232 implementation with **Hardware Flow Control disabled.** Use pins 2, 3, and 5 for **Received Data**, **Transmitted Data**, and **Ground** respectively.
- Use a Null-Modem type cable (cross cable) when communicating with a PC.

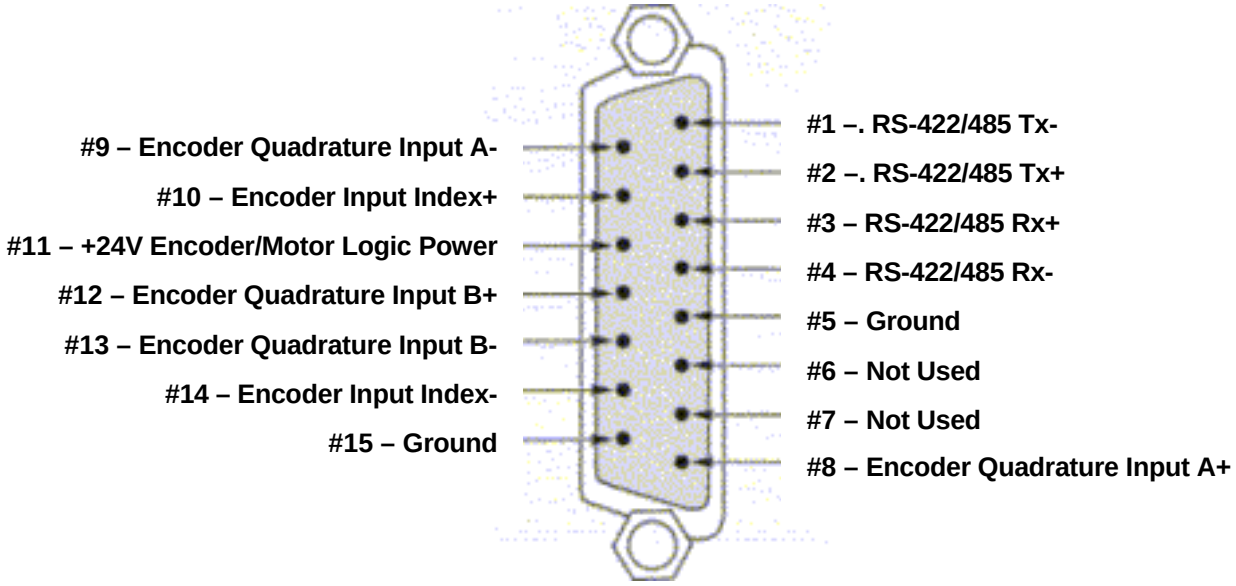
11. COM2 Connector



NOTES:

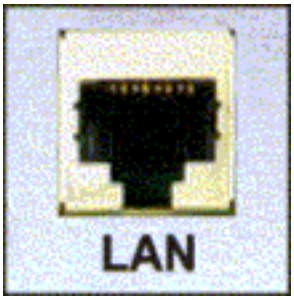
- The unit communicates via a simplified 3-wire RS-232 implementation with **Hardware Flow Control disabled.** Use pins 2, 3, and 5 for **Received Data**, **Transmitted Data**, and **Ground** respectively. Pin 9 provides **+5V** to power a remote control pendant.
- Use a Null-Modem type cable (cross cable) when communicating with a PC.

12. COM3 / Mark on the Fly Connector



NOTE: The maximum current for the **+24V Encoder/Motor Logic Power** Supply (Pin #11) is 800mA.

13. LAN Connector



Standard LAN connector. Use for connecting to a Remote API or *WinLase* software.

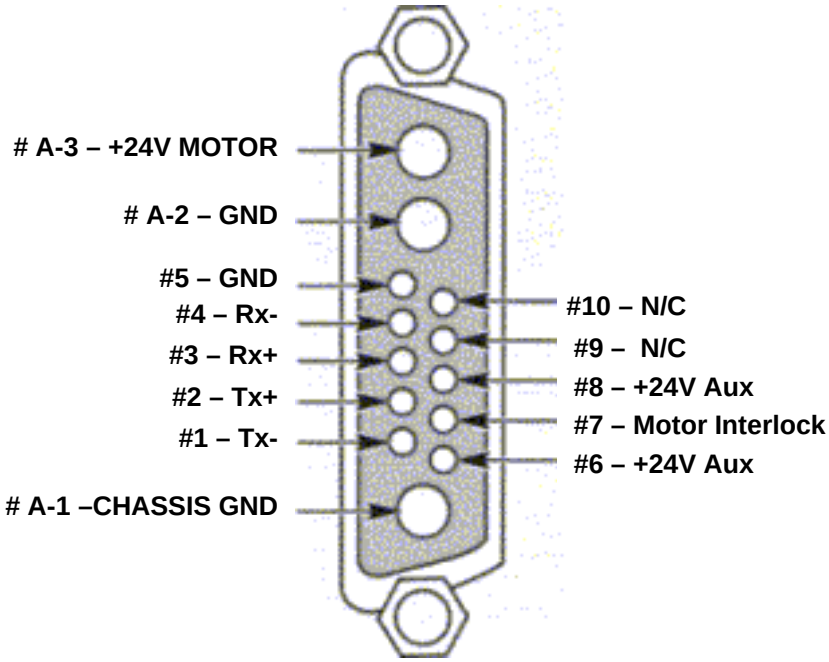
14. USB Connector

A single Type A USB Port is available on the back of the unit for adding additional storage space using a USB Flash Drive, or "thumb drive". Not all flash drives are supported. One suggested brand and model is the *PNY Optima Pro Attache 2GB*. Do ***not*** connect any USB device other than a USB flash drive to these ports.

15. SERVICE Connector

Reserved for factory diagnostic use.

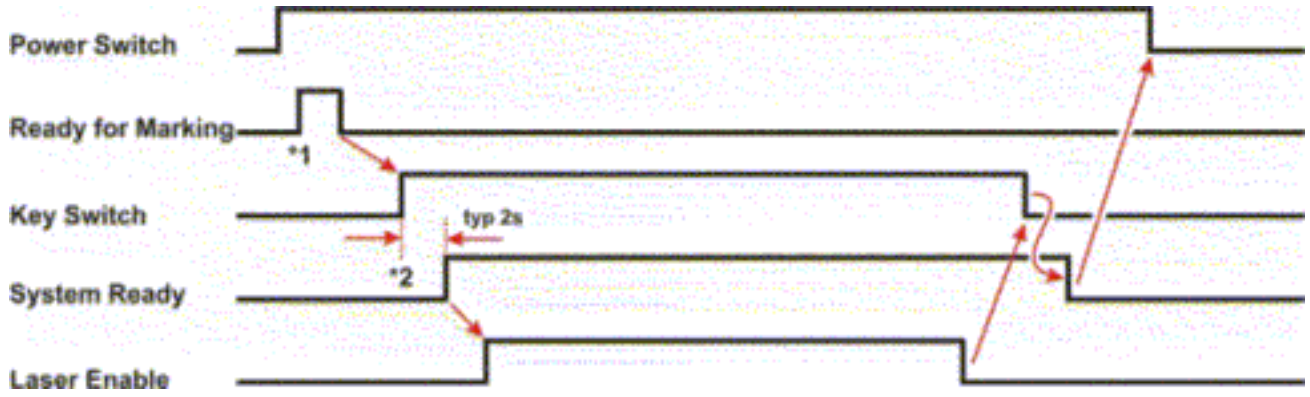
16 -19 Motor #1 - 4



Motor Connector 13W3 Female	
PIN NO.	**DESCRIPTION**
A-3	+24V MOTORS (Controlled by E-Stop circuit)
A-2	POWER SUPPLY GROUND
1	Motor RS-422 Tx-
2	Motor RS-422 Tx+
3	Motor RS-422 Rx+
4	Motor RS-422 Rx-
5	Power Supply Ground
6	+24V Aux (Always On)
7	Motor Interlock
8	+24V Aux (Always On)
9	No Connection
10	No Connection
A-1	CHASSIS GROUND

Section II. Timing Diagrams

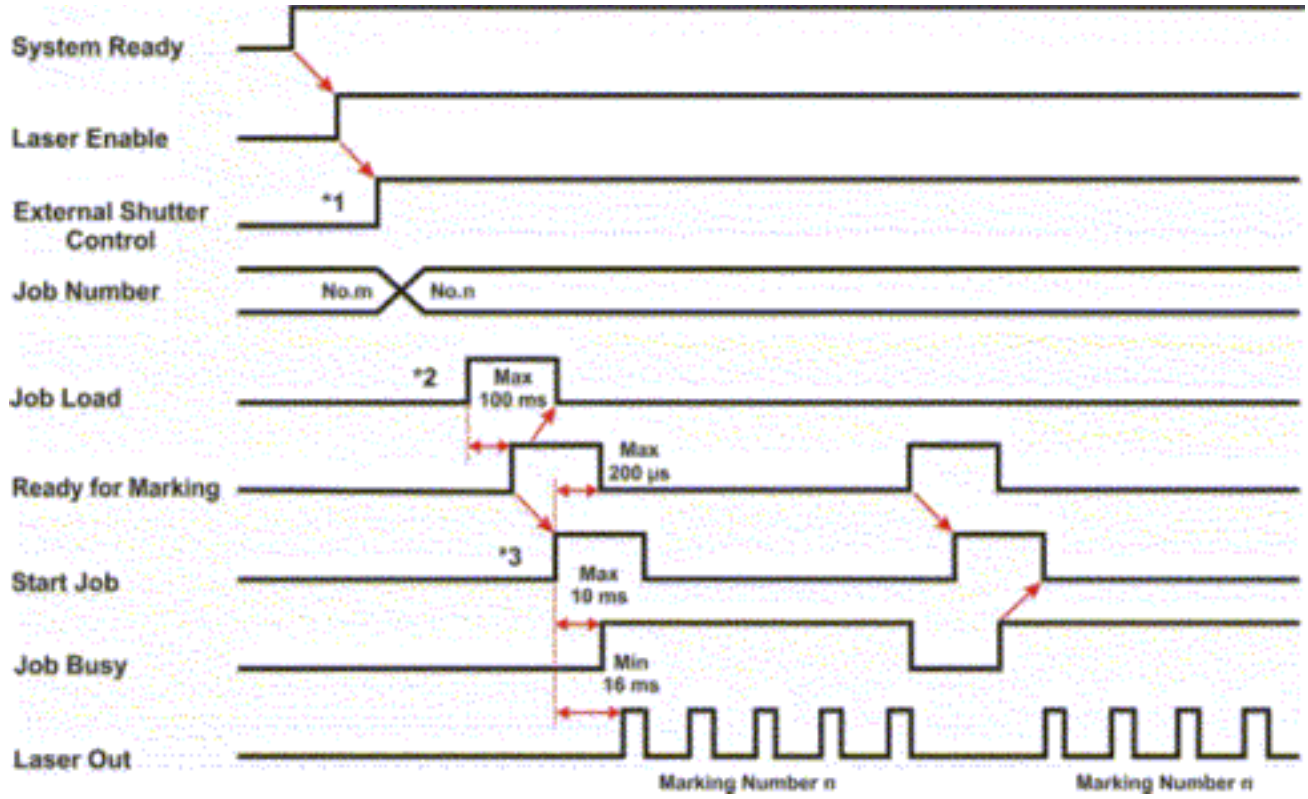
Power ON / OFF



*1 After turning the **Power Switch** ON, the **Ready for Marking** signal turns ON for several seconds as part of the self-check routine. It can take up to 30 seconds to boot the marker control card. Once remote API or WinLase streaming access is available it is possible to proceed.

*2 The **System Ready** turns ON two seconds after the Key switch turns ON.

Normal Operation (Local Execution Mode)

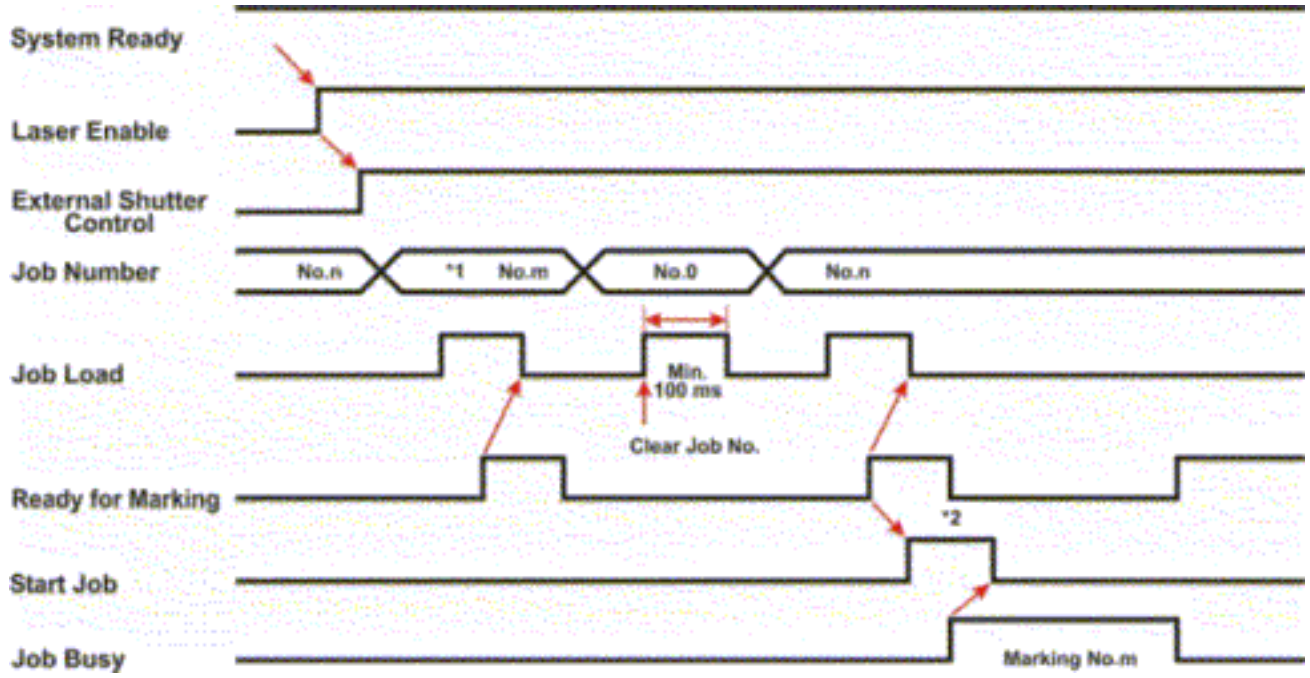


*1 It is necessary to turn the **External Shutter Control** ON for laser marking. When the **External Shutter Control** is OFF, the laser will not fire because the safety shutter is closed. ***You must wait 100ms after opening the shutter before sending a start job command.***

*2 If cache mode is not enabled in the WinLase I/O Job Load mode, it may be necessary to wait several seconds for **Job Load** to turn ON.

*3 The **Start Job** (Start Mark) should be input once the **System Ready** is ON, the **External Shutter Control** has been ON for at least 100ms, the **Laser Enable** is ON, and the **Ready for Marking** is ON. If the **Start Job** turns ON and **Laser Enable** is OFF, the laser does not fire. **Start Job** can be configured to trigger on rising edge, sinking edge, high level, or low level. The system will fault if **Start Job** is sent within 100ms of **External Shutter Control**.

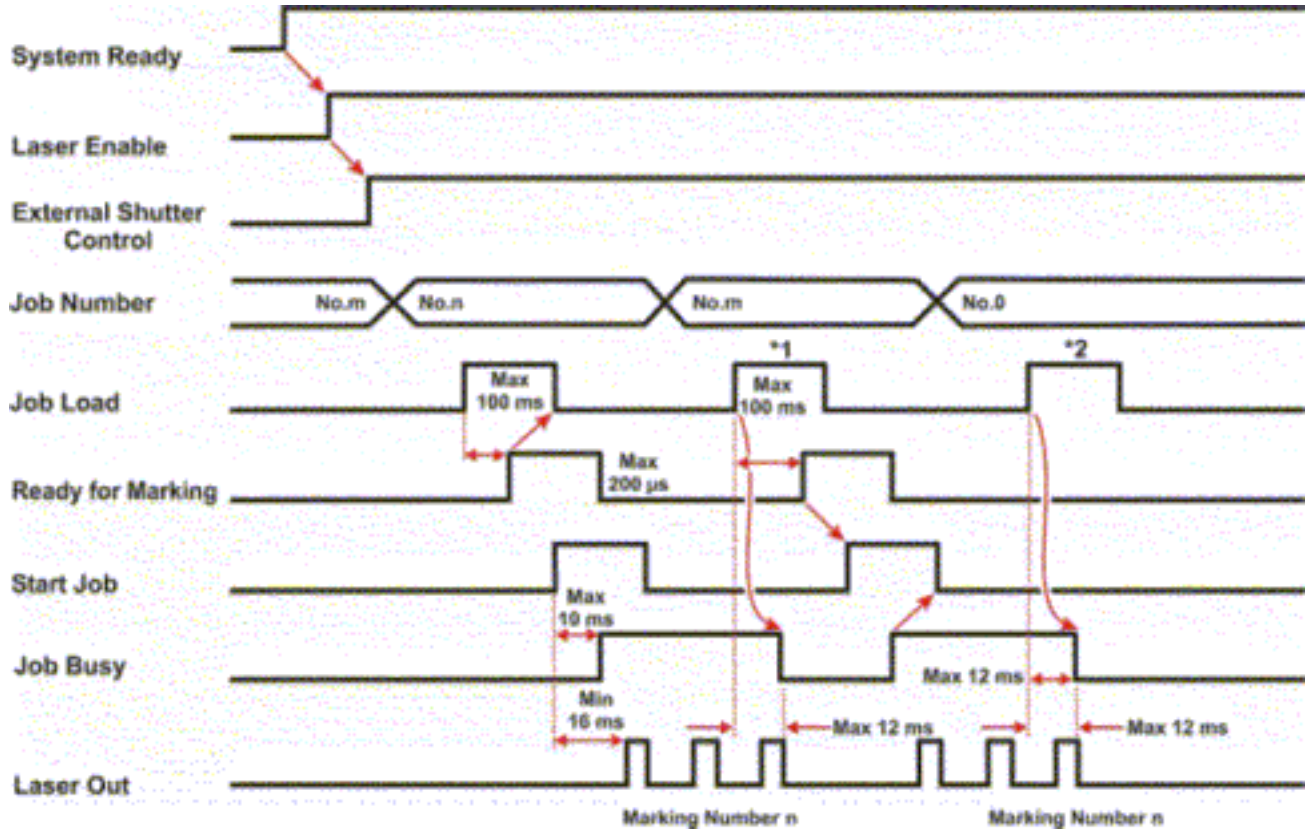
Job Select (Local Execution Mode)



*1 **Job Number** is an 8-bit binary number equivalent to integers between 1 and 255. If you set the binary number to zero, indicating **Job No.0**, and toggle the **Job Load** bit ON, the **Job Number** is cleared. (**Ready for Marking** turns OFF or remains OFF). In addition, if you set a Job No. that has not been downloaded into the unit, and toggle ON the Job Load, **Ready for Marking** will turn OFF or remain OFF.

*2 Toggle **Start Job** ON after **Ready for Marking** turns ON to start the cycle.

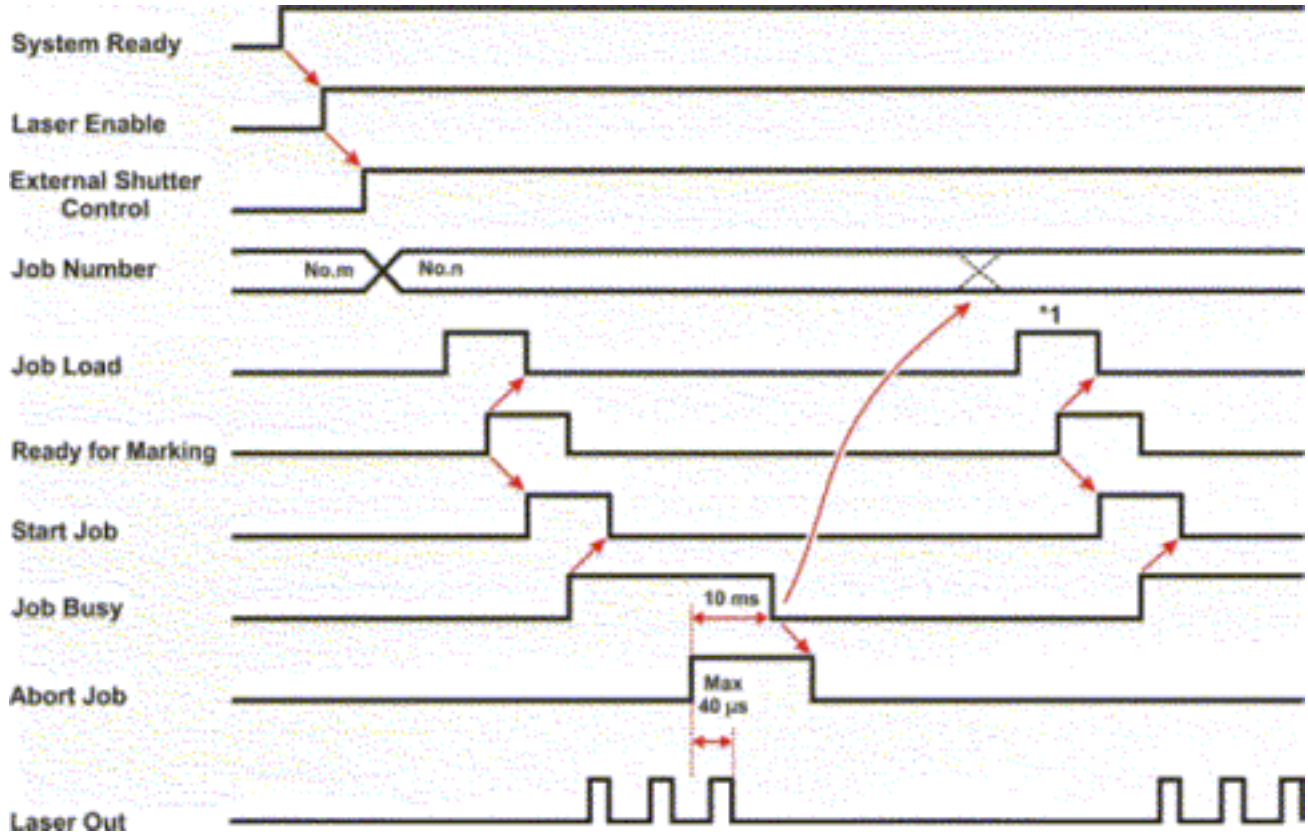
Job Select (During Marking)



*1 When you set a **Job Number** that exists in unit memory and toggle **Job Load** ON during a mark cycle, marking execution stops. In addition, the Job No. changes to the selected one. After the **Ready for Marking** turns ON, then when the **Start Job** turns ON, the marking execute with the selected **Job Number**.

*2 When you set the **Job No. 0**, and toggle ON the **Job Load**, the marking execution stops. Then the **Job Number** is cleared. (However the **Ready for Marking** does not turn ON). In addition, if you select a **Job Number** that has not been previously downloaded into the unit, and toggle the **Job Load** ON, the **Job Number** is cleared too and **Ready for Marking** will turn OFF or remain OFF. If you want to start marking again, you have to select a valid job number and toggle the **Job Load** ON.

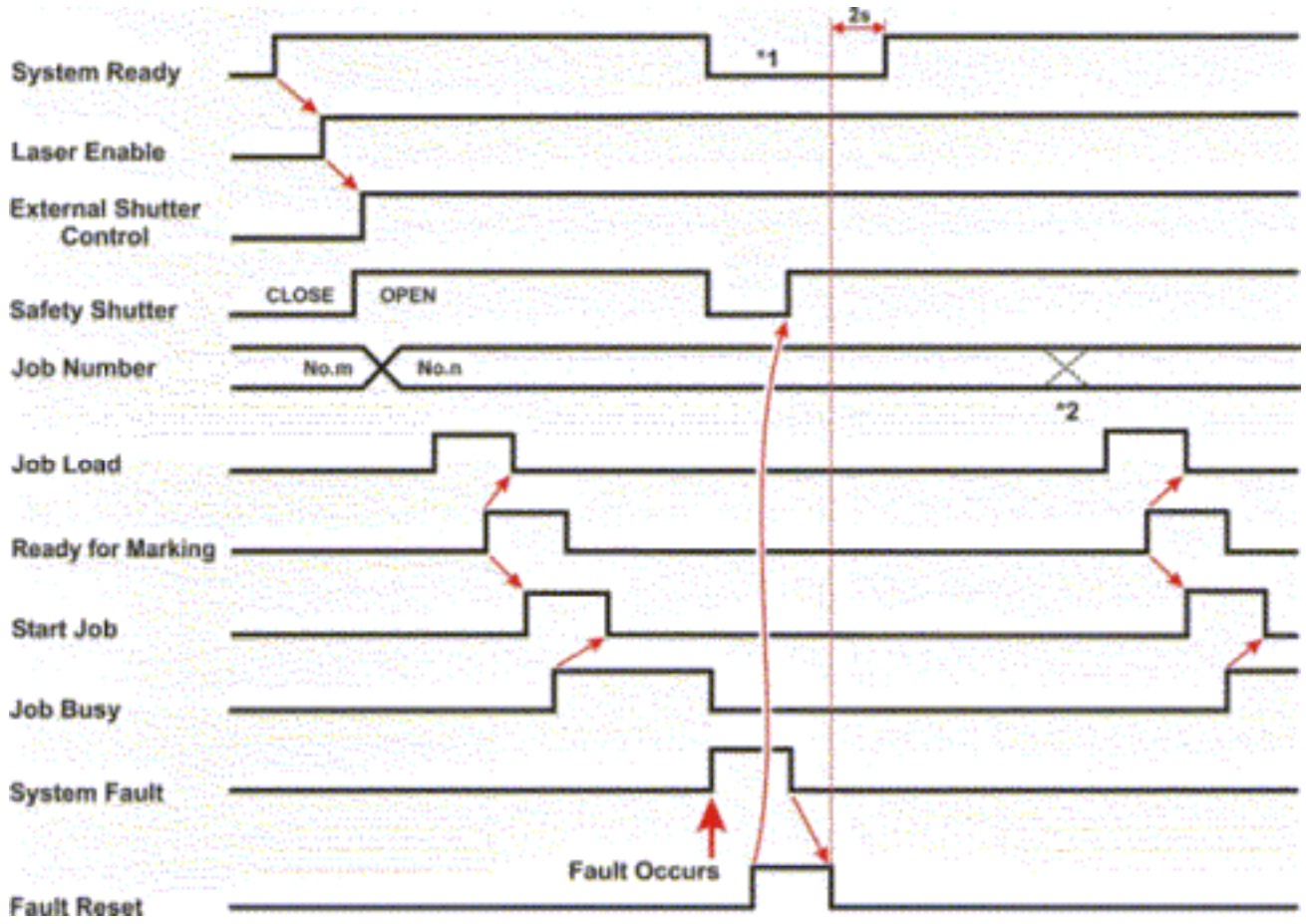
JOB ABORT – Requires WinLase Interlock Configuration



*1 After you input the **Abort Job**, if you want to start marking again, you have to set the **Job Number** and toggle the **Job Load** ON. In addition, if **Abort Job** turns ON not during marking execution, you still need to set the **Job Number** and toggle the **Job Load** ON

*2 After inputting **Abort Job**, the next marking process starts from the beginning of the job. (It does not continue the previous marking job.)

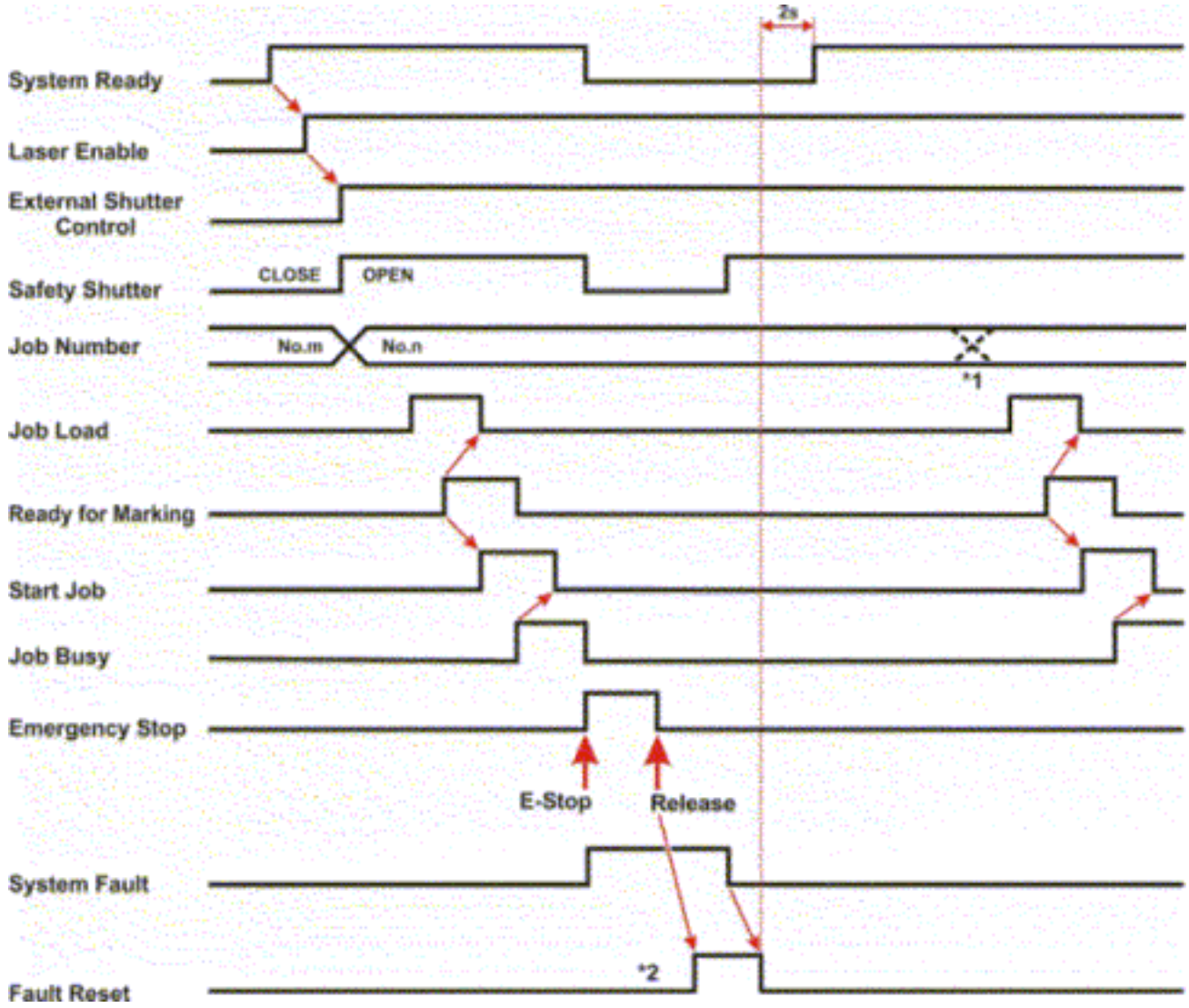
SYSTEM FAULT



*1 When the unit is in a fault condition, the **System Fault** is turned ON and the **System Ready** is turned OFF. Then when the **System Fault** is cleared using **Fault Reset,** (and Fault Reset is OFF), **System Ready** is turned ON 2 seconds later.

*2 After inputting **Fault Reset**, if you want to start marking again, you have to again set the **Job Number** and toggle **Job Load** ON.

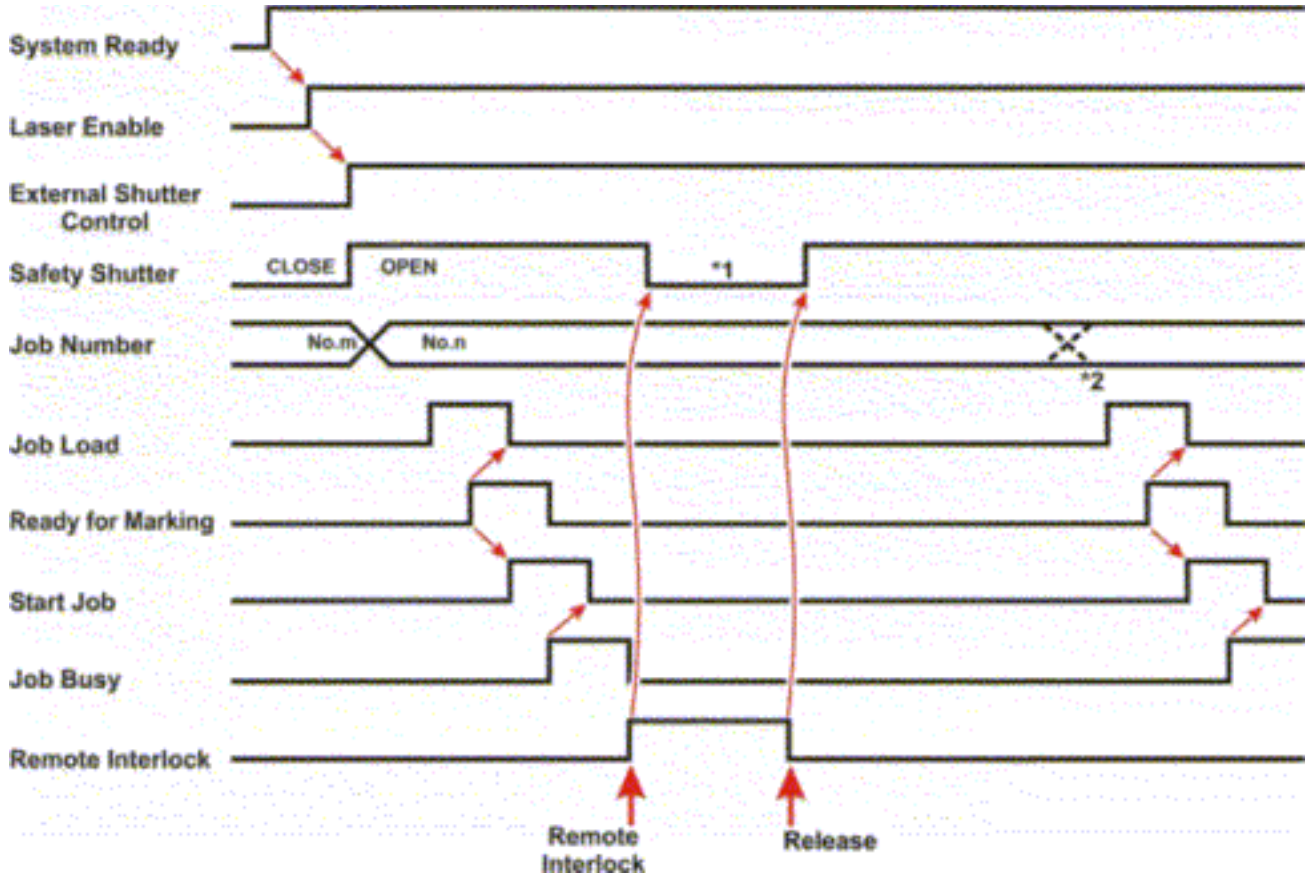
EMERGENCY STOP



*1 After the Fault Reset is turned ON, in order to restart a job you need to set the job number and to turn the **Job Load** ON.

*2 After the Emergency Stop is released and the **Fault Reset** is toggled ON, or the key switch is cycled OFF then ON, the **System Fault** turns OFF. Once the **Fault Reset** is turned OFF the **System Ready** turns ON 2 seconds later. It is necessary to re-load the job by resetting the **Job Number** and toggling **Job Load** ON briefly to continue.

REMOTE INTERLOCK



*1 The **Safety Shutter** closes while the **Remote Interlock** is open.

*2 After the **Remote Interlock** is closed, set the **Job Number** and toggle the **Job Load** to ON.

APPENDIX C
LMF HP PULSED FIBER LASER REFERENCE MATERIAL

Section I: Pulsed Laser Characteristics for –HP and -SM models

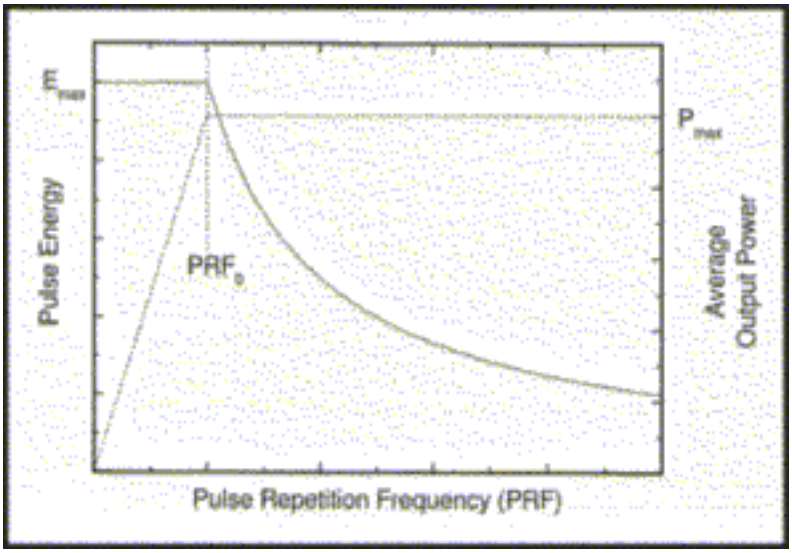
Miyachi Unitek –HP model laser markers use a laser engine that offers extremely fine control of laser parameters. In addition to power, speed, and frequency it is also possible to control select the laser pulse width independent of the frequency chosen. Laser output characteristics including detailed documentation on the selectable pulse widths are described in this chapter. These settings are described as “Waveforms” or “Waveform Mode” to indicate that the pulse shape is being controlled independently of frequency.

Non –HP models are Q-switched and pulse width is related to the selected laser pulse frequency.

Pulse Repetition Frequency (PRF_0)

Each waveform has a characteristic frequency at which peak power and pulse energy is optimized. This is called the PRF_0. For maximum peak power applications (like etching of steel) it is especially important that the frequency and peak power correspond to their ideal values.

The tables on pages C-2 and C-3 provide information for LMF70-HP, LMF35-HP, LMF20-HP and LMF20-SM models.



Waveform Reference Table – LMF70-HP and LMF35-HP

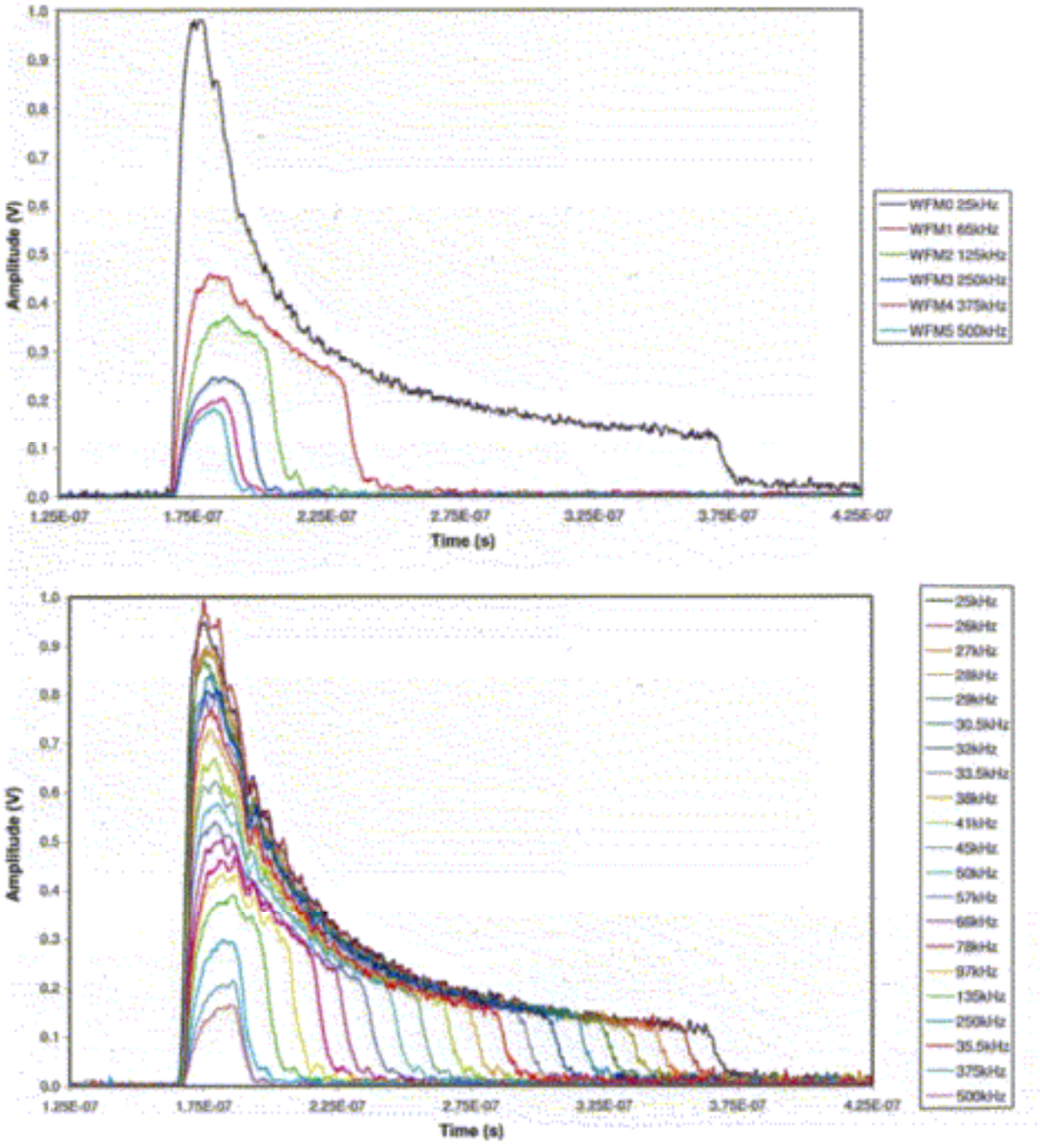
Waveform Number	70W HP Lasers PRF_0 (KHZ) / Emax (mJ)	70W HP Lasers Approximate Electrical Duration (ns)	35W HP Lasers PRF_0 (KHZ) / Emax (mJ)	35W HP Lasers Approximate Electrical Duration (ns)
0	70./.1.0	240	30 / 1.33	250
1	88./.0.87	220	47 / 0.85	130
2	95./.0.76	200	76 / 0.53	60
3	102./.0.71	175	145 / 0.28	30
4	105./.0.69	160	230 / 0.17	20
5	112./.0.64	145	250 / 0.16	9
6	119./.0.61	130	Same as Waveform 5	
7	126./.0.57	120		
8	130./.0.56	115		
9	137./.0.53	105		
10	144./.0.50	100		
11	151./.0.48	90	30 / 1.33	250
12	158./.0.46	80	32 / 1.25	230
13	168./.0.43	65	34 / 1.18	220
14	179./.0.40	58	38 / 1.05	200
15	189./.0.38	60	41 / 0.98	170
16	200./.0.36	55	43.5 / 0.92	150
17	214./.0.34	50	47 / 0.85	130
18	228./.0.32	45	51 / 0.78	110
19	245./.0.29	40	54 / 0.74	100
20	266./.0.27	36	58 / 0.69	90
21	291./.0.25	33	62 / 0.65	80
22	315./.0.23	30	69 / 0.58	70
23	350./.0.21	26	76 / 0.53	60
24	403./.0.18	23	89 / 0.45	50
25	490./.0.15	20	108 / 0.37	40
26	600./.0.12	16	145 / 0.28	30
27	850./.0.08	10	230 / 0.17	20
28	1000./.0.07	10	250 / 0.16	9
29	70./.1.0	270	Same as Waveform 28	
30	70./.1.0	295		
31	70./.1.0	320		
32	70./.1.0	350		
33	70./.1.0	380		
34	70./.1.0	420		
35	70./.1.0	470		
36 — 63	70./.1.0	520		

Waveform Reference Table – LMF20-SM and LMF20-HP

Waveform Number	20W HP Lasers PRF_0 (KHZ) / Emax (mJ)	20W HP Lasers Approximate Electrical Duration (ns)	20W SM Lasers PRF_0 (KHZ) / Emax (mJ)	20W SM Lasers Approximate Electrical Duration (ns)
0	25 / 0.8	200	35 / 0.57	220
1	65 / 0.31	65	55 / 0.36	120
2	125 / 0.16	30	90 / 0.22	55
3	250 / 0.08	15	170 / 0.12	25
4	375 / 0.053	12	270 / 0.074	18
5	500 / 0.04	9	290 / 0.068	15
6	Same as Waveform 5		Same as Waveform 5	
7				
8				
9				
10				
11	25 / 0.8	200	35 / 0.57	220
12	26 / 0.77	190	37 / 0.54	205
13	27 / 0.74	180	39 / 0.51	200
14	28 / 0.71	170	44 / 0.45	190
15	29 / 0.69	160	48 / 0.42	160
16	30.5 / 0.66	150	51 / 0.39	140
17	32/ 0.63	140	55 / 0.36	120
18	33.5 / 0.6	130	60 / 0.33	100
19	35.5 / 0.56	120	63 / 0.32	95
20	38 / 0.53	110	68 / 0.29	85
21	41 / 0.49	100	72 / 0.28	75
22	45 / 0.44	90	80 / 0.25	65
23	50 / 0.4	80	90 / 0.22	55
24	57 / 0.35	70	105 / 0.19	45
25	66 / 0.3	60	125 / 0.16	35
26	78 / 0.26	50	170 / 0.12	25
27	97 / 0.21	40	270 / 0.074	18
28	135 / 0.15	30	290 / 0.068	15
29	250 / 0.08	20	290 / 0.068	15
30 — 63	Same as Waveform 29		Same as Waveform 29	

Example of 20W HP Laser Module Optical Pulse Shapes for various waveform and frequency combinations

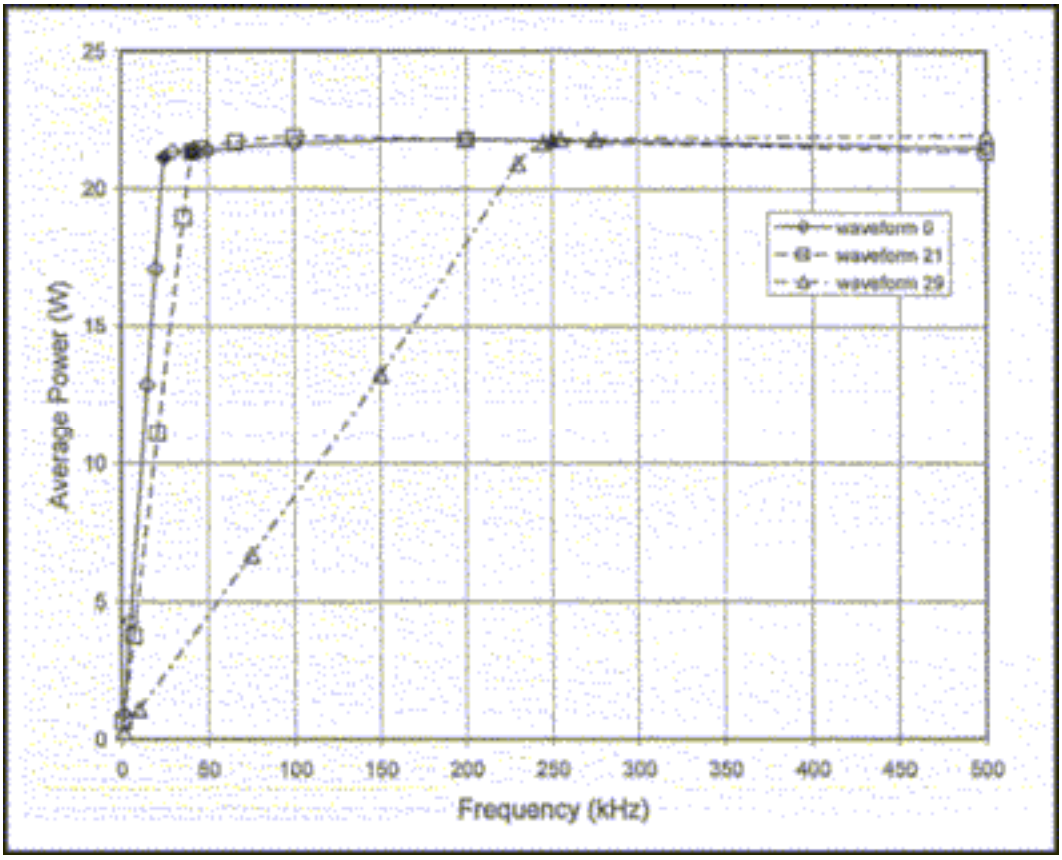
Voltage in the y-axis of these charts is the output from a photodetector diode detecting the output laser pulse and therefore these are analogous to pulse energy over time



TOP: **Waveforms 0 – 5 at PRF_0 for each waveform.**

BOTTOM: **Waveforms 11 – 29 at PRF_0 for each waveform.**

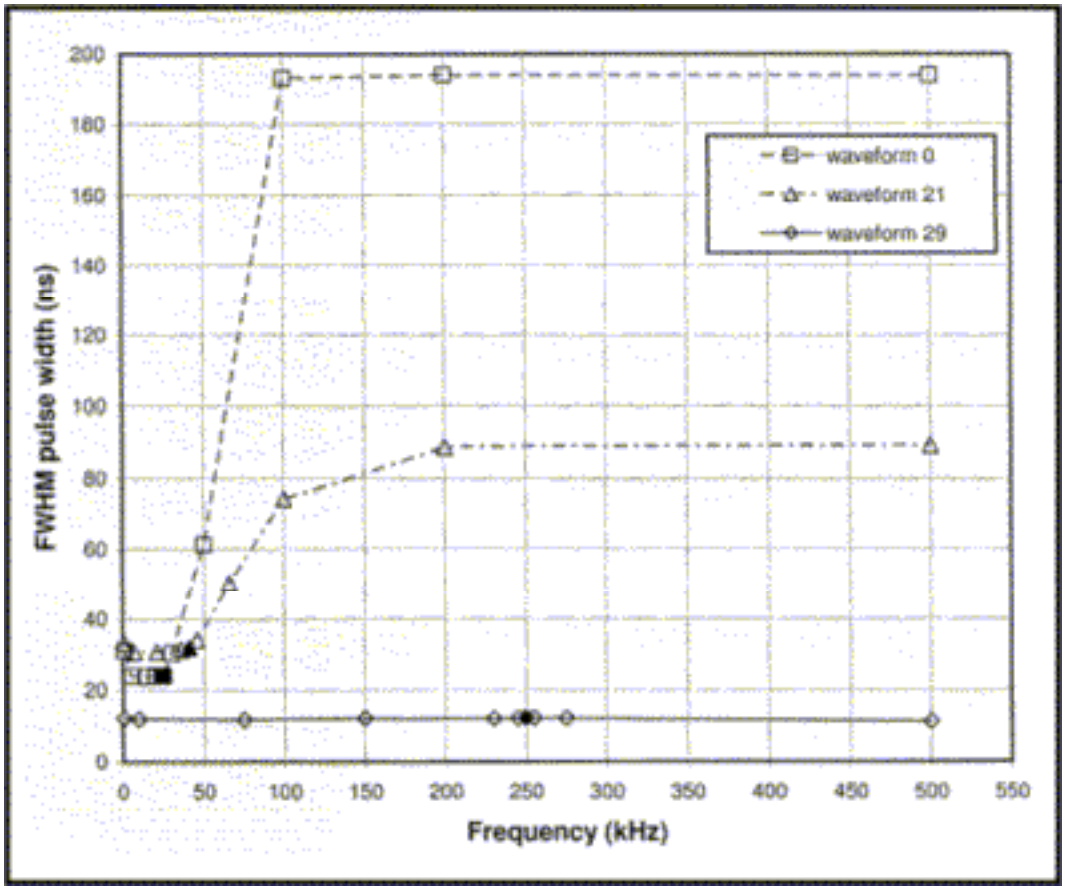
Example of 20W HP Laser Module
Average Power vs. PRF Characteristics



NOTE: Solid points indicate PRF_0.

Waveform 0: PRF_0 = 25 kHz
Waveform 21: PRF_0 = 41 kHz
Waveform 29: PRF_0 = 250 kHz

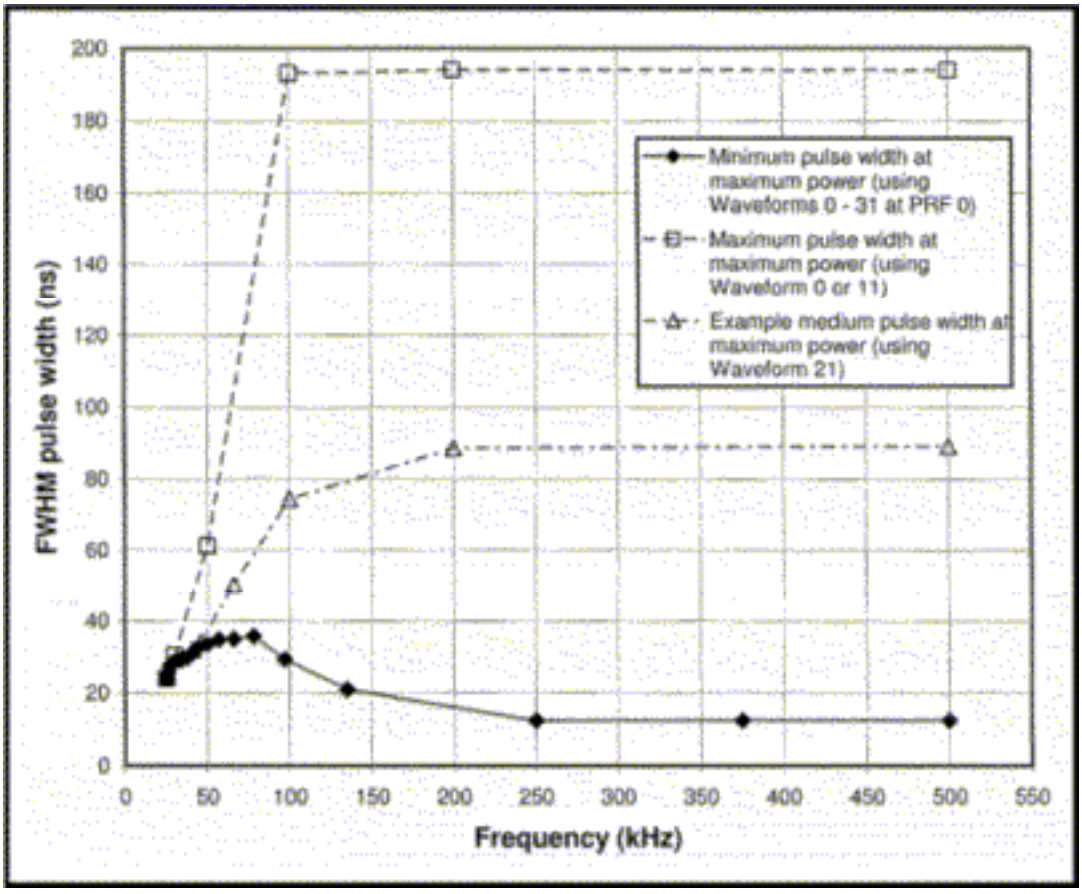
Example of 20W HP Laser Module FWHM Pulse Width vs. PRF Characteristics



NOTE: Solid points indicate PRF_0.

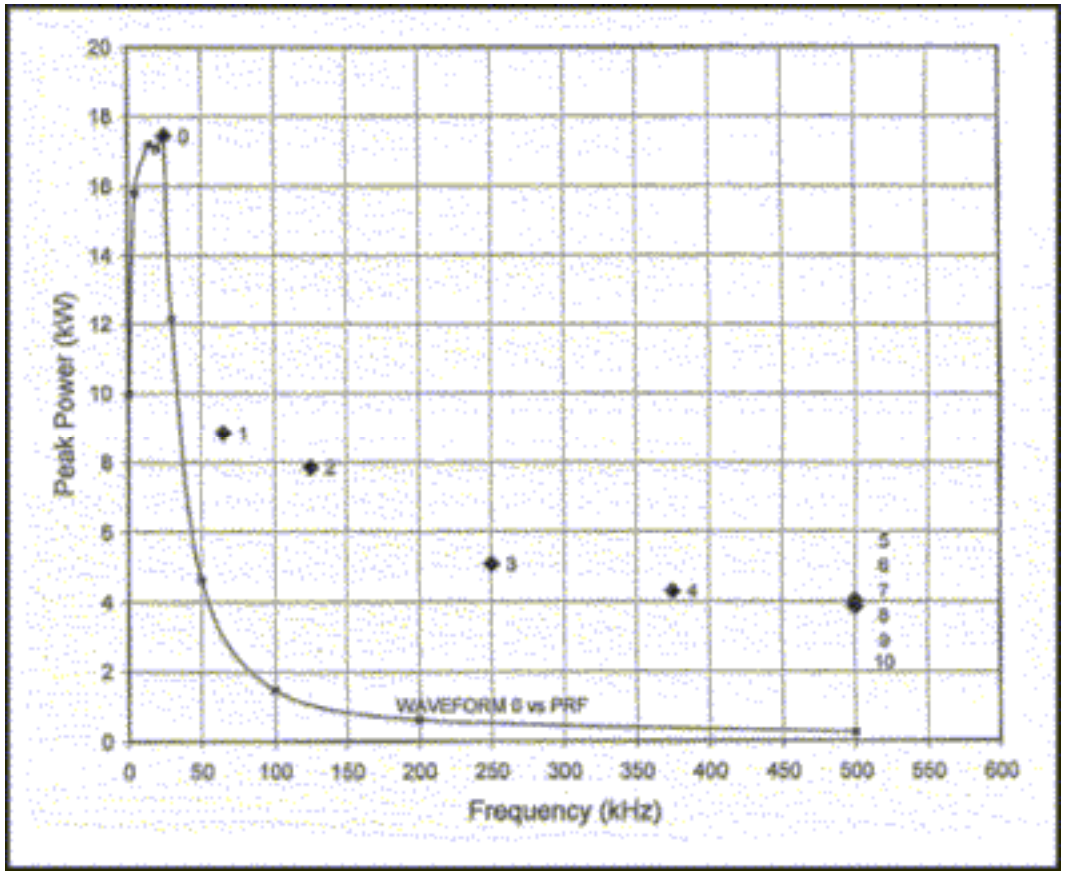
Waveform 0: PRF_0 = 25 kHz
Waveform 21: PRF_0 = 41 kHz
Waveform 29: PRF_0 = 250 kHz

Range Of Achievable FWHM Pulse Width vs. PRF at Maximum Output Power Using Waveforms at, or Above Their Specified PRF_0 Frequency



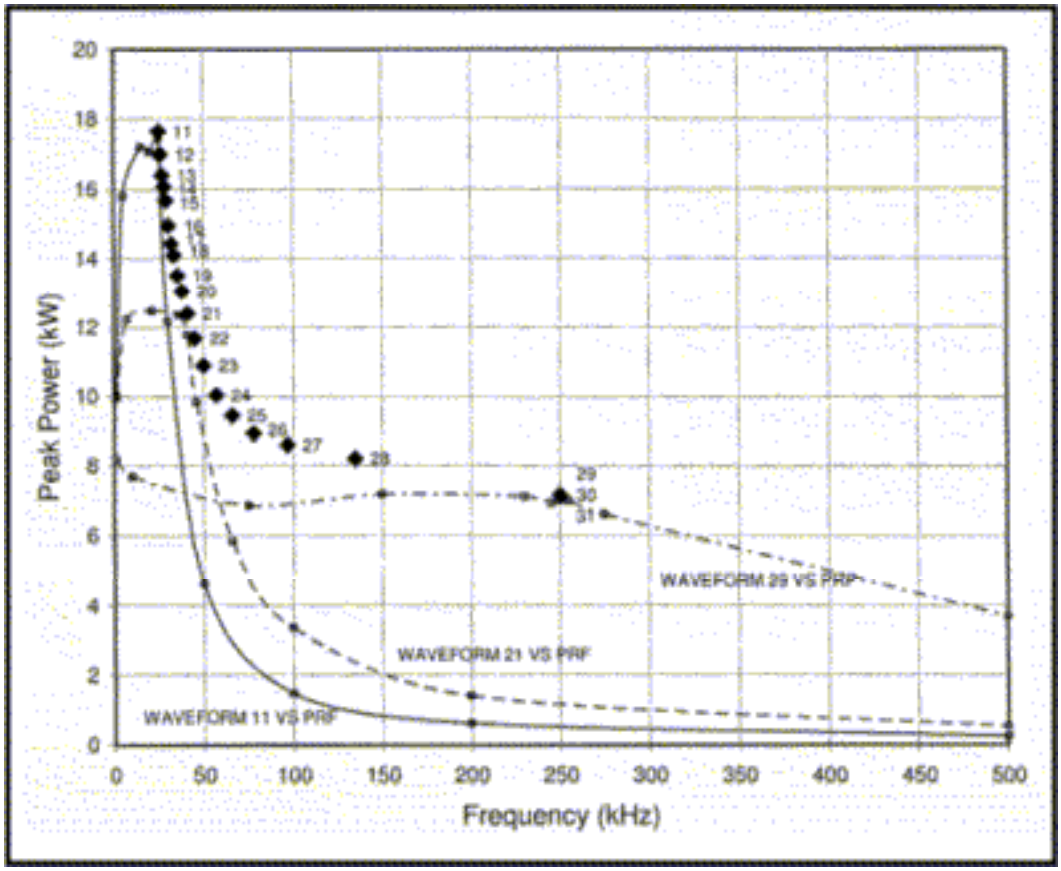
NOTE: This graph does not show operation at pulse rates below PRF_0 for any waveform.

Example of 20W HP Laser Module
Peak Power vs. PRF Characteristics (waveforms 0 – 10)



NOTE: Solid points indicate PRF_0.

Example of HP 20W Laser Module
Peak Power vs. PRF Characteristics (waveforms 11 – 31)



NOTE: Solid points indicate PRF_0.

APPENDIX D
EMBEDDED CONTROLLER REMOTE COMMAND API

Section I. Remote Command API

Remote API Commands (Numerical Listing)

Below is a summary of all API Commands used in the LEC Control PCB (the operational controller used in all LMF Fiber Laser Markers). Immediately following this summary is a detailed list of Remote API Commands (note: the API commands are listed alphabetically).

Value/constant	Description	Firmware Compatibility		
// Control commands //		**2.x**	**6.x**	**7.x**
1	Abort	X	X	X
2	TakeHostControl	X	X	X
3	ReleaseHostControl	X	X	X
4	GetHostControlStatus	X	X	X
5	GetHostInControl	X	X	X
6	EnableBroadcasting	X	X	X
7	LoadHardwareDefaults	X	X	X
8	HardwareReset	X	X	X
9	GetRemoteIP	X	X	X
10	GetKFactor	X	X	X
14	SetPerformanceGlobals	X	X	X
15	ResetPerformanceGlobals	X	X	X
16	OpenCOMPort	X	X	X
17	CloseCOMPort	X*	X*	X*
18	COMWriteLine	X*	X*	X*
19	GoToZ	X	X	X
20	GoToXYZ	X	X	X
21	SetMOTFEncoderRate	X	X	X
22	SetMemBuffer	X	X	X
23	GetMemBuffer	X	X	X
24	GetAvailableRAM	X	X	X
27	COMWriteChar	X*	X*	X*
29	SetUserOutBit	X	X	X

* Command not available in all versions of the specified firmware. See command detail.

Value/constant	Description	Firmware Compatibility		
// **Control commands** // (cont.)		**2.x**	**6.x**	**7.x**
30	GetUserInWord	X	X	X
31	GetAllIOWords	X	X	X
32	SetUserOutInitWord	X	X	X
33	GetUserOutInitWord	X	X	X
34	SampleMOTFEncoderCount	X	X	X
35	ClearMOTFEncoderCount	X	X	X
36	GetMOTFEncoderCount	X	X	X
37	Echo	X	X	X
38	GetLensFileList	X	X	X
39	LoadLensFile	X	X	X
40	SetUserOutPreferences	X	X	X
41	GetUserOutPreferences	X	X	X
42	SetUserOutWord	X	X	X
43	GetVersions	X	X	X
44	GetLaserFileList	X	X	X
45	LoadLaserFile	X	X	X
46	GetMotionFileList		X*	X*
47	LoadMotionFile		X*	X*
48	GetFontFileList	X	X	X
49	GetActiveLaser	X	X	X
50	GetActiveLens	X	X	X
52	GetAvailableDiskSpace	X	X	X
56	ClearCommandCache	X	X	X
57	TurnLaserOn	X	X	X
58	TurnLaserOff	X	X	X
59	GetMotionDeviceNames		X*	X*
60	GetMotionCalFactors		X*	X*
61	SendMotionCommand		X*	X*
63	GetMotionErrorCodes		X*	X*
65	GetMotionStatus		X*	X*
67	GetLastInterlockWord	X	X	X
72	COMWriteCharEx	X*	X*	X*
73	COMWriteLineEx	X*	X*	X*
75	CloseCOMPortEx	X*	X*	X*
76	SetZOffsetRWU		X	X

* Command not available in all versions of the specified firmware. See command detail.

Value/constant	Description	Firmware Compatibility		
// **Object commands** //		**2.x**	**6.x**	**7.x**
100	SetObjectString	X	X	X
102	TransformObject	X	X	X
103	GetObjectRect	X	X	X
104	GetObjectCenter	X	X	X
105	GetObjectType	X	X	X
106	EnableObject	X	X	X
107	GetObjectString	X	X	X
108	GetObjectName	X	X	X
109	SetObjectUserData	X	X	X
110	GetObjectUserData	X	X	X
111	ResetObject	X	X	X
112	ResetUserTransform	X	X	X
113	TransformObjectByName	X	X	X
114	TransformObjectByNameEx	X	X	X
115	SetObjectProfile	X	X	X
116	GetObjectProfile	X	X	X
117	SetObjectProfileFromFile	X	X	X
118	GetObjectNumPasses	X		
119	SetObjectNumPasses	X		
120	GetObjectMarkMode	X		
121	SetObjectMarkMode	X		
122	GetObjectNumMarkingPasses		X	X
123	AddObjectMarkingPass		X	X
124	DeleteObjectMarkingPass		X	X
125	SetObjectPassRepetitions		X	X
126	GetObjectPassRepetitions		X	X
127	TransformObjectNewFill	X	X	X
128	TransformObjectByNameNewFill	X	X	X
136	NewObject			X
137	SetObjectUnicodeString			X
138	GetObjectUnicodeString			X
139	GetObjectVectors			X
140	SetObjectVectors			X
141	RemoveObject			X

Value/constant	Description	Firmware Compatibility		
// Job commands //		**2.x**	**6.x**	**7.x**
200	ClearJobList	X	X	X
201	MakeJobActive	X	X	X
202	RemoveJob	X	X	X
203	GetFlashJobFileList	X	X	X
204	GetUSBJobFileList	X	X	X
205	LoadFlashJob	X	X	X
206	LoadUSBJob	X	X	X
207	ExecuteJobOnce	X	X	X
208	ExecuteJobContinuous	X	X	X
209	GetJobStatus	X	X	X
210	GetLastError	X	X	X
211	GetObjectCount	X	X	X
214	GetJobExecutionStatus	X	X	X
215	SetExternalStartMode	X	X	X
216	GetExternalStartMode	X	X	X
218	GetActiveJob	X	X	X
219	SaveFlashJob	X	X	X
220	SaveUSBJob	X	X	X
221	GetNetworkJobFileList		X	X
222	LoadNetworkJob		X	X
223	SaveNetworkJob		X	X
224	GetLastMotionError		X*	X*
225	NewJob			X

* Command not available in all versions of the specified firmware. See command detail.

Value/constant	Description	Firmware Compatibility		
// **Administration** //		**2.x**	**6.x**	**7.x**
500	SetAdminPIN	X	X	X
501	GetAdminPIN	X	X	X
502	SetDHCPMode	X	X	X
503	GetDHCPMode	X	X	X
504	SetLocalGateway	X	X	X
505	GetLocalGateway	X	X	X
506	SetLocalIP	X	X	X
507	GetLocalIP	X	X	X
508	SetNodeFriendlyName	X	X	X
509	GetNodeFriendlyName	X	X	X
510	SetSubnetMask	X	X	X
511	GetSubnetMask	X	X	X
512	SetUserPIN	X	X	X
513	GetUserPIN	X	X	X
514	SetCOMPortSpeed	X*	X*	X*
515	GetCOMPortSpeed	X*	X*	X*
516	SetCOMPortAssignments	X	X	X
517	GetCOMPortAssignments	X	X	X
518	SetLocalTime	X	X	X
519	GetLocalTime	X	X	X
523	ConnectNetworkShare		X	X
524	SetCOMPortSpeedEx	X*	X*	X*
525	GetCOMPortSpeedEx	X*	X*	X*
526	GetLocalDeviceList	X	X	X
527	SetActiveLocalDevice	X	X	X
528	SetCOMPortMode		X*	X*
529	GetCOMPortMode		X*	X*

* Command not available in all versions of the specified firmware. See command detail.

Remote API Commands - Detail (Alphabetical Order)

A detail of the Remote API Commands are list alphabetically below by function name.

Abort	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Stops the execution of a job.
Implementation:	"**1**" or "1,blocking"
Parameters:	blocking: The blocking behavior of the command. 0 = do not block, 1 = block Data type: int
Returns:	An API Response code
Comments:	During Abort processing, there are two tasks that the LEC performs. The first task is to immediately stop the execution of an in-process job. The second task is to clear all queued data and wait for a handshake from the FIFO to complete the abort process. When using the blocking option, the Abort command waits until both tasks are complete before returning. When the non-blocking option is used, the Abort command returns immediately after the first task. When calling "1", the shorter syntax version of the command, this will use the blocking option. The longer syntax version is available on firmware 2.2.27.4 and higher and 6.0.1.35 and higher.
Also see:	None

AddObjectMarkingPass	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Add a *MarkingPass* to a marking object contained in the ActiveJob with the specified vector list.
Implementation:	"**123**,objectindex,vectorlist,repetitions,markspeed,jumpspeed,jumpdelay,markdelay,polydelay,laserpower,laseroffdelay,laserondelay,zaxis,frequency,pulsewidth,eightbitword,varijumpdelay,varijumplength,wobbleamplitude,wobblefrequency,powerreset,varipolydelay,powerrampstart,powerrampend,powerrampstartrate,powerrampendrate,powerrampendtime"
Parameters:	objectindex: The zero-based index of the object to change. Data type: int Valid range: [0 to (object count – 1)]
	vectorlist: The vector list profile to set. 0 = outline list, 1 = fill list. Data type: int
	repetitions: The number of times to repeat this *MarkingPass* when executing the object. Data type: int Valid range: [> 0]
Returns:	*0,newpassindex* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. A marking object can have an unlimited number of *MarkingPasses* for both outline vectors and fill vectors. Usually a *MarkingPass* represents a *Profile*. Each *MarkingPass* can also have a number of repetitions. See SetObjectProfile for a definition of the Profile parameters included in this command.
Also see:	*SetObjectProfile, GetObjectProfile, GetObjectNumMarkingPasses, DeleteObjectMarkingPass, SetObjectPassRepetitions, GetObjectPassRepetitions*

ClearCommandCache	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Provides a way to clear all list operations that have already been cached in the command FIFO, while the LEC is actively waiting for a Start Mark signal.
Implementation:	"**56**"
Parameters:	None
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must have been started in *Cache* mode, or the call will return *NotInCacheMode*. The system must be actively waiting for a Start Mark signal for this command to succeed, or *NotWaitingForStartMark* will be returned. After a successful call to *ClearCommandCache*, the command FIFO will be cleared, but the LEC will still be waiting for a Start Mark signal. Providing a Start Mark signal at this time will switch the *Ready* output to LOW, allowing the LEC to cache additional commands. If additional commands are cached in the FIFO before the pending Start Mark is cleared, two (2) Start Mark signals will be required to execute the additional commands. The *Ready* output will switch LOW after the first Start Mark signal.
Also see:	*Abort*

ClearJobList	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Removes all loaded jobs from memory.
Implementation:	"**200**"
Parameters:	None
Returns:	An API Response code
Comments:	The *ActiveJob* is cleared when this call completes.
Also see:	*RemoveJob, LoadUSBJob, LoadFlashJob*

ClearMOTFEncoderCount	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Clears the MOTF encoder counter to zero.
Implementation:	"**35**"
Parameters:	None
Returns:	An API response code
Comments:	The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. The LEC must have a valid Advanced (MOTF) license.
Also see:	*SampleMOTFEncoderCount, GetMOTFEncoderCount, GetJobStatus*

CloseCOMPort	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (up to firmware versions 2.2.27.4, 6.0.1.36 and 7.0.1.36 respectively)**
Purpose:	Closes the COM port opened with the *OpenCOMPort* command. On Firmware 6.x and 7.x LEC devices, this API has been deprecated and users should use *CloseCOMPortEx* instead.
Implementation:	"**17**"
Parameters:	None
Returns:	An API Response code
Comments:	Client must call *OpenCOMPort* with the appropriate port number before making this call.
Also see:	*OpenCOMPort, COMWriteLine, COMWriteChar*

CloseCOMPortEx	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (from firmware versions 2.2.27.5, 6.0.1.37 and 7.0.1.37 respectively)**
Purpose:	Closes the specified COM port opened with the *OpenCOMPort* command.
Implementation:	"**75**,portnum"
Parameters:	portnum: The COM port to close. Valid range: [1,2,3]
Returns:	An API Response code
Comments:	Client must call *OpenCOMPort* with the appropriate port number before making this call.
Also see:	*OpenCOMPort, COMWriteLineEx, COMWriteCharEx*

COMWriteChar	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (up to firmware versions 2.2.27.4, 6.0.1.36 and 7.0.1.36 respectively)**
Purpose:	Writes a single character to the COM port, which was opened with the *OpenCOMPort* command. This API has been deprecated and users should use COMWriteCharEx instead.
Implementation:	"**27**,timeout,char"
Parameters:	timeout: The time, in ms, to wait for a response to the character sent to the COM port. If no response is expected, use the value 0 (zero).
	char: The character to send to the COM port.
Returns:	*response:* The response received from the COM connected device. If the port times out, *PortTimeout* is returned. If *portnum* is invalid, *WrongPortNumber* is returned. If a value of 0 (zero) is specified for *timeout*, *Success* is returned.
Comments:	Client must call *OpenCOMPort* before making this call. When using *COMWriteChar*, a single character is sent to the COM connected device, without an appended line feed. The LEC will then wait for a single character response from the COM connected device. If the device responds with more than one character, only the first character in the response is returned. If no character is detected, the command will timeout.
Also see:	*OpenCOMPort, CloseCOMPort, COMWriteLine*

COMWriteCharEx	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (from firmware versions 2.2.27.5, 6.0.1.37 and 7.0.1.37 respectively)**
Purpose:	Writes a single character to the specified COM port, which was opened with the *OpenCOMPort* command.
Implementation:	"**72**,portnum,timeout,char"
Parameters:	portnum: The COM port to use. Valid range: [1,2,3]
	timeout: The time, in ms, to wait for a response to the character sent to the COM port. If no response is expected, use the value 0 (zero).
	char: The character to send to the COM port.
Returns:	*response:* The response received from the COM connected device. If the port times out, *PortTimeout* is returned. If *portnum* is invalid, *WrongPortNumber* is returned. If a value of 0 (zero) is specified for *timeout*, *Success* is returned.
Comments:	Client must call *OpenCOMPort* before making this call. When using *COMWriteCharEx*, a single character is sent to the COM connected device, without an appended line feed. The LEC will then wait for a single character response from the COM connected device. If the device responds with more than one character, only the first character in the response is returned. If no character is detected, the command will timeout.
Also see:	*OpenCOMPort, CloseCOMPort, COMWriteLine*

COMWriteLine	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (up to firmware versions 2.2.27.4, 6.0.1.36 and 7.0.1.36 respectively)**
Purpose:	Writes a string of characters to the COM port opened with the *OpenCOMPort* command. This API has been deprecated and users should use COMWriteLineEx instead.
Implementation:	"**18**,timeout,string"
Parameters:	timeout: The time in ms to wait for a response to the string sent to the COM port. If no response is expected, use the value 0 (zero).
	string: The string of characters to send to the COM port. The LEC will append a line feed character to the end of the string.
Returns:	*responsecode,response:* The response received from the COM connected device pre-pended by an API Response code. If the port times out, *PortTimeout* is returned. If *portnum* is invalid, *WrongPortNumber* is returned. If a value of 0 (zero) is specified for *timeout*, *Success* is returned. Example return string: 0,hello
Comments:	Client must call the *OpenCOMPort* before making this call. When using *COMWriteLine*, the string sent to the device is appended with a line feed. The LEC will then wait for a response from the COM connected device, which must have an appended line feed. If no line feed is detected, the command will timeout.
Also see:	*OpenCOMPort, CloseCOMPort, COMWriteChar.*

COMWriteLineEx	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (from firmware versions 2.2.27.5, 6.0.1.37 and 7.0.1.37 respectively)**
Purpose:	Writes a string of characters to the specified COM port opened with the *OpenCOMPort* command.
Implementation:	"**73**,portnum,timeout,string"
Parameters:	portnum: The COM port to use. Valid range: [1,2,3]
	timeout: The time in ms to wait for a response to the string sent to the COM port. If no response is expected, use the value 0 (zero).
	string: The string of characters to send to the COM port. The LEC will append a line feed character to the end of the string.
Returns:	*responsecode,response:* The response received from the COM connected device pre-pended by an API Response code. If the port times out, *PortTimeout* is returned. If *portnum* is invalid, *WrongPortNumber* is returned. If a value of 0 (zero) is specified for *timeout*, *Success* is returned. Example return string: 0,hello
Comments:	Client must call the *OpenCOMPort* before making this call. When using *COMWriteLineEx*, the string sent to the device is appended with a line feed. The LEC will then wait for a response from the COM connected device, which must have an appended line feed. If no line feed is detected, the command will timeout.
Also see:	*OpenCOMPort, CloseCOMPort, COMWriteChar.*

ConnectNetworkShare	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Makes a connection to a network resource.
Implementation:	"**523**,remotesharename,username,password"
Parameters:	remotesharename: Specifies the network resource to connect to Valid range: [256 characters maximum]
	username: Specifies the user name for making the connection Valid range: [49 characters maximum]
	password: Specifies the password to be used when making the connection Valid range: [63 characters maximum]
Returns:	If the command fails because of a network connectivity error (NetworkConnectFail), the response is in the form *33,extendederrorinfo* where *33 NetworkConnectFail* and *extendederrorinfo* is the return value from the Windows API call *WNetAddConnection3*. For example, if the call fails because of an Access_Denied error, the response will be 33,5. Please refer to the Windows documentation for more details on the *extendederrorinfo* value. All other responses will be an API Response code.
Comments:	Client must call *TakeHostControl* before making this call. This command supports only the "Microsoft Windows Network" provider. The share cannot be located directly on a Domain Controller. The *remotesharename* must be in the format \\server\share. For example, to connect to a share called Production on a machine named server 01, use "\\server01\Production" without the quotes.
Also see:	None

DeleteObjectMarkingPass	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Remove an existing *MarkingPass* from the marking object contained in the ActiveJob referencing the specified vector list.
Implementation:	"**124**,objectindex,vectorlist,passindex"
Parameters:	objectindex: The zero-based index of the object to change. Data type: int Valid Range: [0 to (object count – 1)]
	vectorlist: The vector list profile to set. 0 = outline list, 1 = fill list. Data type: int
	passindex: The index value of the MarkingPass to delete. Data type: int Valid Range: [0 to (numpasses – 1)]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. A marking object can have an unlimited number of *MarkingPasses* for both outline vectors and fill vectors. Usually a *MarkingPass* represents a *Profile*. Each *MarkingPass* can also have a number of repetitions. This command will fail if there is only one *MarkingPass* left in the list. Each object must have at least one *MarkingPass*.
Also see:	*SetObjectProfile, GetObjectProfile, GetObjectNumMarkingPasses, AddObjectMarkingPass, SetObjectPassRepetitions, GetObjectPassRepetitions*

Echo	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Echo a string
Implementation:	"**37**,string"
Parameters	string: A string to send that will be echoed back by the Remote API: Valid range: [3000 characters]
Returns:	string: The string value is sent.
Comments:	This command can be used to verify communications and for keep-alive purposes.
Also see:	None

EnableBroadcasting	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Enables or disables the broadcast of messages by the Host.
Implementation:	"**6**,state"
Parameters	state: The state of the Broadcast engine: 0 = idle 1 = broadcasting messages Valid range: [0,1]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Some high speed operations, especially when using the RS-232 port, may be briefly interrupted during a broadcast cycle. In these cases, the client can disable broadcasting. Note that when broadcasts are disabled, the LEC will no longer be visible on the local network, but will continue to have full API functionality.
Also see:	None

EnableObject	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Enable / disable execution of the specified object.
Implementation:	"**106**,objectindex,stste"
Parameters	objectindex: The zero-based index of the object to change. Valid range: [0 to (object count – 1)]
	state: The enabled state of the object: 0 = The object will not execute when the job is executed 1 = The object will execute when the job is executed (default) Valid range: [0,1]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. This call can be used to turn on / off the execution of the specified object.
Also see:	*TakeHostControl, GetJobStatus*

ExecuteJobContinuous	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Starts the execution of the Active job and will execute the job in an infinite loop.
Implementation:	"**208**,cacheobjects"
Parameters	cacheobjects: Flag indicating whether to cache objects in the command FIFO. 0 = do not cache objects, but wait for the StartMark signal. 1 = cache objects immediately 2 = cache all instructions in hardware FIFO (from firmware versions 2.2.27.5, 6.0.1.37 and 7.0.1.37 respectively) Valid range: [0,1]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be set to a job currently loaded in RAM with a call to *MakeJobActive*. The LEC will wait to start marking until a hardware *StartMark* signal is received, and will continue repeating this process until *Abort* is called. The cacheobjects flag effects how the list of marking objects is processed and sent to the command FIFO. If *cacheobjects* = 0, the system will wait for the StartMark signal before sending the objects in the list to the command FIFO. This option is useful when using External Control, etc. If *cacheobjects* = 1, all objects in the job are committed to the command FIFO (up to the memory capacity of the FIFO) as soon as the *ExecuteJobContinuous* command is received, or the previous cycle has completed. This option is useful when very high speed repetitive performance is required, and the data may change between each mark (serialization, transforms, etc.). If *cacheobjects* = 2, all objects in the job are committed to the hardware FIFO as soon as the *ExecuteJobContinuous* command is received. Since all commands are now in the hardware, variable marking data is not possible. This option is useful when very high speed repetitive performance is required, such as in a mark on the fly application with very high speed marks. This command returns a response immediately. If an Interlock has been signaled, the return value is *ResetInterlock*. The Interlock condition must be cleared, and a call made to *Abort* to clear the internal Interlock flag. To determine if the ActiveJob is loading, use the *GetJobStatus* command. To determine if the ActiveJob is executing, use the *GetJobExecutionStatus* command.
Also see:	*TakeHostControl, MakeJobActive, GetJobStatus, GetJobExecutionStatus, Abort, LoadFlashJob, LoadUSBJob*

ExecuteJobOnce	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Starts the execution of the ActiveJob, and runs it once without repeat.
Implementation:	"**207**,cacheobjects"
Parameters	cacheobjects: Flag indicating whether to cache objects in the command FIFO. 0 = do not cache objects, but wait for the StartMark signal. 1 = cache objects immediately Valid range: [0, 1]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be set to a job currently loaded in RAM with a call to *MakeJobActive*. The LEC will wait to start marking until a hardware StartMark signal is received. The job can be stopped by calling *Abort* at any time. The *cacheobjects* flag effects how the list of marking objects is processed and sent to the command FIFO. If *cacheobjects* = 0, the system will wait for the StartMark signal before sending the objects in the list to the command FIFO. This option is useful when using External Control, etc. If *cacheobjects* = 1, all objects in the job are committed to the command FIFO (up to the memory capacity of the FIFO) as soon as the *ExecuteJobOnce* command is received. This option is useful when very high speed repetitive performance is required. This command returns a response immediately. If an Interlock has been signaled, the return value is *ResetInterlock*. The Interlock condition must be cleared, and a call made to *Abort* to clear the internal Interlock flag. To determine if the ActiveJob is loading, use the *GetJobStatus* command. To determine if the ActiveJob is executing, use the *GetJobExecutionStatus* command.
Also see:	*TakeHostControl, GetJobStatus, GetJobExecutionStatus, Abort, LoadFlashJob, LoadUSBJob*

GetActiveJob	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the name and index value of the currently ActiveJob.
Implementation:	"**218**"
Parameters	None
Returns:	*0,value,name* if there was no error, where: *value* = the I/O Job Selection index of the job *name* = the name of the job or *errorcode* if there was an error. *errorcode* is an API response code
Comments:	Before interacting with a job, it must be made active with the *MakeJobActive* command.
Also see:	*LoadFlashJob, LoadUSBJob, MakeJobActive*

GetActiveLaser	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the name of the currently active laser driver file.
Implementation:	"**49**"
Parameters	None
Returns:	*0,laserfile* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	None
Also see:	*GetLaserFileList, LoadLaserFile*

GetActiveLens	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the name of the currently active lens correction table.
Implementation:	"**50**"
Parameters	None
Returns:	*0,lensfile* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	None
Also see:	*GetLensFileList, LoadLensFile*

GetAdminPIN	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current Administration PIN.
Implementation:	"**501**"
Parameters	None
Returns:	*adminpin*: A string representing the AdminPIN.
Comments:	The *AdminPIN* is used with the pendant interface, and provides password protection for Administrative functions.
Also see:	*SetAdminPIN, GetUserPIN, SetUserPIN*

GetAllIOWords	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the state of all digital inputs and outputs as two WORDS.
Implementation:	"**31**"
Parameters	None
Returns:	*StandardWORD,ExtendedWORD*: An 18-bit WORD representing the Standard inputs and outputs followed by a 32-bit WORD representing the Extended I/O card inputs and outputs. The two WORDS are separated by a comma. <u>*StandardWORD* bit definitions:</u> Bit 0: User In 1 — Bit 10: User Out 1 Bit 1: User In 2 — Bit 11: User Out 2 Bit 2: User In 3 — Bit 12: User Out 3 Bit3: User In 4 — Bit 13: User Out 4 Bit 4: Start Mark — Bit 14: Mark in Progress Bit5: Job Load — Bit 15: Job Busy Bit 6: Interlock 1 — Bit 16: System Error Bit 7: Interlock 2 — Bit 17: Ready Bit 8: Interlock 3 Bit 9: Interlock 4

GetAllIOWords (cont.)	
Returns (cont.):	*ExtendedWORD* bit definitions: Bit 0: User In 5 — Bit 16: User Out 5 Bit 1: User In 6 — Bit 17: User Out 6 Bit 2: User In 7 — Bit 18: User Out 7 Bit3: User In 8 — Bit 19: User Out 8 Bit 4: User In 9 — Bit 20: User Out 9 Bit5: User In 10 — Bit 21: User Out 10 Bit 6: User In 11 — Bit 22: User Out 11 Bit 7: User In 12 — Bit 23: User Out 12 Bit 8: User In 13 — Bit 24: User Out 13 Bit 9: User In 14 — Bit 25: User Out 14 Bit 10 User In 15 — Bit 26: User Out 15 Bit 11 User In 16 — Bit 27: User Out 16 Bit 12 User In 17 — Bit 28: User Out 17 Bit 13 User In 18 — Bit 29: User Out 18 Bit 14 User In 19 — Bit 30: User Out 19 Bit 15 User In 20 — Bit 31: User Out 20
Comments:	Bit 0 is defined as the LSB of the WORD. A bit value of 0 indicates the input is LOW, a value of 1 indicates the input is HIGH. ***Note:*** 8-75 Markers support User In bits 1-4 only, 8-77 Markers support User In bit 1-12 only, where bits 5-12 are defined in Appendix B as "Job Select Bit 1-8" All 8-75 and 8-77 Markers support User Out bits 1-4 only.
Also see:	*SetUserOutBit, GetUserInWord*

GetAvailableDiskSpace	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the amount of free Flash and USB space.
Implementation:	"**52**"
Parameters	None
Returns:	*0,flashspace,usbspace* if there was no error, where: *flashspace* = Total free space in the system Flash, in kB. *usbspace* = Total free space on an attached USB drive, in kB. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	If no USB drive is detected, *usbspace* will be 0.
Also see:	None

GetAvailableRAM	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the amount of available RAM from the Windows CE operating system.
Implementation:	"**24**"
Parameters	None
Returns:	*memorycount*: The amount of available RAM as reported by the Windows CE operating system.
Comments:	None
Also see:	None

GetCOMPortAssignments	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the COM port to be used for the available interfaces. On Firmware 6.x and 7.x LEC devices, this API has been deprecated and users should use *GetCOMPortMode* instead.
Implementation:	"**517**"
Parameters	None
Returns:	*assignments*: Comma-delimited string representing the assignments. The string is in three sections: Section 1 = Pendant assignment Section 2 = Remote API assignment Section 3 = Motion control assignment Example: "1,2,3" would indicate that the Pendant is assigned to COM1, the Remote API to COM2 and Motion control to COM3. A zero in any location indicates there is no COM port assigned to that interface.
Comments:	Client must call *TakeHostControl* before making this call. There are three COM ports available, COM1, COM2 and COM3.
Also see:	*SetCOMPortAssignments, SetCOMPortSpeed, GetCOMPortSpeed*

GetCOMPortMode	**Supported Platforms: Firmware Code 6.x and 7.x (from firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Gets the interfaces that have been mapped to the COM ports.
Implementation:	"**529**"
Parameters	None
Returns:	*modes*: Comma-delimited string representing the interface mode of each COM port. The string is in three-sections containing the interface mode of each COM port: Section 1 = COM1 mode Section 2 = COM2 mode Section 3 = COM3 mode Where the interface mode is defined as: 0 = unassigned 1 = Motion interface 2 = Remote API interface 3 = Pendant interface Example: "0,1,2" would indicate that COM1 is unassigned, COM2 will expose the Motion interface and COM3 will expose the Remote API interface.
Comments:	Client must call *TakeHostControl* before making this call. The LEC device must be restarted for these changes to take effect. There are three COM ports available, COM1, COM2, and COM3.
Also see:	*SetCOMPortMode, SetCOMPortSpeedEx, GetCOMPortSpeedEx*

GetCOMPortSpeed	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (up to firmware versions 2.2.27.4, 6.0.1.36 and 7.0.1.36 respectively)**
Purpose:	Get the baud rates of the interfaces that use the COM ports. This API has been deprecated and users should use *GetCOMPortSpeedEx* instead.
Implementation:	"**515**"
Parameters	None
Returns:	*speeds*: A comma-delimited string representing the speeds assigned to the interfaces. The string is in three sections: Section 1 = Pendant port baud rate Section 2 = Remote API port baud rate Section 3 = Motion controller port baud rate
Comments:	Client must call *TakeHostControl* before making this call. There are three COM ports available on the LEC: COM1, COM2 and COM3. Only the baud rate can be configured on the ports. The remaining settings are: 8 data bits No parity 1 stop bit XON/XOFF (Software handshake)
Also see:	*SetCOMPortSpeed, SetCOMPortAssignments, GetCOMPortAssignments*

GetCOMPortSpeedEx	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (from firmware versions 2.2.27.4, 6.0.1.37 and 7.0.1.37 respectively)**
Purpose:	Get the current baud rates of the COM ports.
Implementation:	"**525**"
Parameters	None
Returns:	*speeds*: A comma-delimited string representing the speeds assigned to the COM ports. The string is in three sections: Section 1 = COM1 baud rate Section 2 = COM2 baud rate Section 3 = COM3 baud rate
Comments:	Client must call *TakeHostControl* before making this call. There are three COM ports available on the LEC: COM1, COM2 and COM3. Only the baud rate can be configured on the ports. The remaining settings are: 8 data bits No parity 1 stop bit No hardware control
Also see:	*SetCOMPortSpeedEx, SetCOMPortMode, GetCOMPortMode*

GetDHCPMode	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current DHCP mode.
Implementation:	"**503**"
Parameters	None
Returns:	"StaticIP": LEC is in fixed IP address mode. "Autodetect": LEC is in Dynamic IP address mode.
Comments:	If the LEC is in DHCP mode, at power up it will attempt to get an IP address from a DHCP server on the local network.
Also see:	*SetDHCPMode*

GetExternalStartMode	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current ExternalStart mode of the ActiveJob.
Implementation:	"**216**"
Parameters	None
Returns:	*0,mode* if there was no error, where mode: 0 = When port is HIGH 1 = When port is LOW 2 = After transition from LOW > HIGH 3 = After transition from HIGH > LOW or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. The External Start mode controls what type of signal transition on the Start Mark input will trigger the start of job execution.
Also see:	*SetExternalStartMode*

GetFlashJobFileList	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets a comma-delimited list of all jobs stored in Flash memory
Implementation:	"**203**"
Parameters	None
Returns:	*0,joblist* if there was no error, where: *joblist* = A comma-delimited list of all jobs stored in Flash or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Flash memory is the LEC internal storage memory.
Also see:	*GetUSBJobFileList*

GetFontFileList	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets a comma-delimited list of all font files stored in Flash memory
Implementation:	"**48**"
Parameters	None
Returns:	*0,fontfile* where: *fontfile* = The name of the font file stored in Flash. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Flash memory is the LEC internal storage memory.
Also see:	None

GetHostControlStatus	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Indicates whether the Client currently has exclusive control of the LEC
Implementation:	"**4**"
Parameters	None
Returns:	"In Control": Client currently has exclusive control of the LEC. "NotInControl": Client currently does not have exclusive control of the LEC.
Comments:	None
Also see:	*TakeHostControl, ReleaseHostControl*

GetHostInControl	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current host interface that has exclusive control of the LEC.
Implementation:	"**5**"
Parameters	None
Returns:	"HostPendant": The pendant has exclusive control. "HostLANStream": The LAN based streaming interface (WinLase) has exclusive control. "HostGUI": A User interface running on the LEC has exclusive control. "HostBluetooth": The Bluetooth interface has exclusive control. "HostLan": The LAN interface has exclusive control. "HostRS232": The RS-232 interface has exclusive control. "HostLocalIO": The LocalIO job loading Host has exclusive control.
Comments:	None
Also see:	*TakeHostControl, Release HostControl*

GetJobExecutionStatus	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the command execution status of the FIFO.
Implementation:	"**214**"
Parameters	None
Returns:	"JobIdle": The command FIFO is not currently executing commands. "Busy": The command FIFO is busy executing commands.
Comments:	Use this command to check the execution status of the command FIFO. If the status is *JobIdle*, the command FIFO is not currently executing commands. There may be commands being loaded into the command FIFO, however. To check the loading status, use *GetJobStatus*. If the status is *Busy*, the command FIFO is currently executing commands. It may be possible to load commands into the FIFO while it is executing. To set the ActiveJob to a job currently loaded into RAM call *MakeJobActive*.
Also see:	*GetJobStatus, MakeJobActive, LoadUSBJob, LoadFlashJob, ExecuteJobOnce, ExecuteJobContinuous*

GetJobStatus	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the command loading status of the job engine.
Implementation:	"**209**"
Parameters	None
Returns:	"JobIdle": There are no commands waiting to be loaded into the command FIFO. "Busy": The job engine is currently loading commands into the command FIFO.
Comments:	Use this command to check the status of the job engine, which feeds the command FIFO. If the status is *JobIdle*, the job engine is available for loading new commands into the FIFO. The FIFO may still be actively executing commands, however. To check the execution status, use *GetJobExecutionStatus*. If the status is *Busy*, the job engine is currently loading commands into the command FIFO, and is not available. To set the ActiveJob to a job currently loaded into RAM call *MakeJobActive*.
Also see:	*GetJobExecutionStatus, MakeJobActive, LoadUSBJob, LoadFlashJob, ExecuteJobOnce, ExecuteJobContinuous*

GetKFactor	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the calibration factor of the lens configuration.
Implementation:	"**10**"
Parameters	None
Returns:	*kfactor*: The calibration factor of the lens configuration, in field units (bits/mm).
Comments:	Use this command to discover the conversion between real world units and field units. Note: Firmware versions 6.x and 7.x maintain native coordinates in µm.
Also see:	*SetAlignmentGlobals*

GetLaserFileList	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets a comma-delimited list of all laser driver files, and their display names, stored in Flash memory.
Implementation:	"**44**"
Parameters	None
Returns:	*0,laserfile1,lasername1,laserfileN,lasernameN* a comma-delimited list where: *laserfile* = The name of the laser driver stored in Flash. *lasername* = The name of the laser driver as it appears in WinLase LAN. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Flash memory is the LEC internal storage memory.
Also see:	*LoadLaserFile*

GetLastError	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the formatted last error string from the server.
Implementation:	"**210**"
Parameters	None
Returns:	*errorstring*: The last server error represented as a string. The string is in the format: *errorcode,objectname* where: *objectname* is the name of the object that generated the run time error and errorcode is: Firmware 2.x an API Response code Firmware 6.x/7.x an API Response code or a System Error code.
Comments:	An application should periodically check *GetLaserError*, as some errors may be generated asynchronously after *Success* was returned from a previous command. For example, after a (potential) long marking process has been started with a call to *ExecuteJobContinuous*, an error could occur that is generated after the call to *ExecuteJobContinuous returns*. The error is cleared on the next call to *ExecuteJobOnce* or *ExecuteJobContinuous*.
Also see:	*ExecuteJobOnce, ExecuteJobContinuous*

GetLastInterlockWord	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets a WORD that represents the interlocks input values that triggered an interlock state.
Implementation:	"**67**"
Parameters	None
Returns:	*word*: The last interlock state when an interlock condition was triggered. Bit 1 represents Interlock 1, Bit 2 represents Interlock 2, etc.
Comments:	An application can check the states the interlocks were in when the LEC entered the Interlock latched state.
Also see:	None

GetLastMotionError	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.3 and 7.0.2.3 respectively)**
Purpose:	Gets the formatted last motion error string from the server.
Implementation:	"**224**"
Parameters	None
Returns:	*0,axisname,xerror,xerrorstring,yerror,yerrorstring*, a comma-delimited list where: *axisname* = The axis that generated the error. *xerror* = The x-axis error code. Refer to motor manufacturer for error code definitions. *xerrorstring* = The x-axis extended error information or 0 (zero) if there is no data. *yerror* = The y-axis error code. Refer to motor manufacturer for error code definitions. *yerrorstring* = The y-axis extended error information or 0 (zero) if there is no date. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	An application should periodically check *GetLastMotionError*, as some errors may be generated asynchronously after *Success* was returned from a previous command. For example, after a (potentially) long marking process has been started with a call to *ExecuteJobContinuous*, an error could occur that is generated after the call to *ExecuteJobContinuous* returns.
Also see:	*ExecuteJobOnce, ExecuteJobContinuous*

GetLensFileList	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets a comma-delimited list of all lens (correction table) files, and their display names, stored in Flash memory.
Implementation:	"**38**"
Parameters	None
Returns:	*0,lensfile1,lensname1,lensfileN,lensnameN,* a comma-delimited list where: *lensfile* = The name of the correction table stored in Flash *lensname* = The name of the correction table as it appears in WinLase. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Flash memory is the LEC internal storage memory.
Also see:	*LoadLensFile*

GetLocalDeviceList	**Supported Platforms: WinXP (SP3), Win7 (Pro), Win8/8.1 (Pro)**
Purpose:	Gets a list of device ID's that represent LEC devices that have been detected on the local PC.
Implementation:	"**526**"
Parameters	None
Returns:	*devicelist*: A comma-delimited list of local device ID's.
Comments:	None
Also see:	*SetActiveLocalDevice*

GetLocalGateway	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the default local Gateway address for the LEC.
Implementation:	"**505**"
Parameters	None
Returns:	*gateway*: The Gateway address in dot notation format. Ex: 192.168.42.1
Comments:	None
Also see:	*SetLocalGateway*

GetLocalIP	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the local IP address for the LEC
Implementation:	"**507**"
Parameters	None
Returns:	*ipaddress*: The IP address in dot notation format. Ex: 192.168.42.1
Comments:	None
Also see:	SetLocalIP

GetLocalTime	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current local time and date
Implementation:	"**519**"
Parameters	None
Returns:	*localtime*: A comma-delimited string containing the local time data. The string is in the format: year,month,day,hour,minute,second
Comments:	None
Also see:	*SetLocalTime*

GetMemBuffer	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the internal memory buffer at the specified index.
Implementation:	"**23**,bufferindex"
Parameters	bufferindex: The index of the memory buffer to retrieve Valid range: [1 to 10]
Returns:	*bufferstring*: The contents of the specified memory buffer
Comments:	A string based object (barcodes and text) can be configured to retrieve the next string to mark from an internal memory buffer at execution time. There are 10 buffers available. The string object must be configured to use these memory buffers.
Also see:	*SetMemBuffe*r

GetMOTFEncoderCount	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current value of the MOTF encoder counter.
Implementation:	"**36**"
Parameters	None
Returns:	*0,count* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	The value count is in units of field bits. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJobStatus, use the *GetJobStatus* command. The LEC must have a valid Advanced (MOTF) license.
Also see:	*GetJobStatus, SampleMOTFEncoderCount, ClearMOTFEncoderCount*

GetMotionCalFactors	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Gets the x-axis (and optionally y-axis) calibration factors for the specified motion device.
Implementation:	"**60**,devicename"
Parameters	devicename: The name of the motion device
Returns:	*0,xfactor,yfactor*, a comma-delimited list where: *xfactor* = The calibration factor of the x-axis *yfactor* = The calibration factor of the y-axis or *errorcode* if there was an error. *errorcode* is an API response code. Linear motion device units: micro-steps / mm Rotary motion device units: micro-steps / degree
Comments:	Use this command to discover the conversion between real world units and motor steps.
Also see:	*GetMotionDeviceNames*

GetMotionDeviceNames	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Gets a comma-delimited list of the motion devices contained in the currently loaded motion configuration file.
Implementation:	"**59**"
Parameters	None
Returns:	*0,motiondevice1,motiondevice2,motiondeviceN* or *errorcode* if there was an error. *errorcode* is an API response code
Comments:	The motion device name is used in other motion calls
Also see:	*GetMotionErrorCodes, GetMotionCalFactors, SendMotionCommand*

GetMotionErrorCodes	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Gets the x-axis (and optionally y-axis) error codes for the specified motion device by reading the device directly.
Implementation:	"**63**,devicename"
Parameters	devicename: The name of the motion device.
Returns:	*0,axisname,xerror,xerrorstring,yerror,yerrorstring*, a comma-delimited list where: *axisname* = The axis that generated the error. *xerror* = The x-axis error code. Refer to motor manufacturer for error code definitions. *xerrorstring* = The x-axis extended error information or 0 (zero) if there is no data. *yerror* = The y-axis error code. Refer to motor manufacturer for error code definitions. *yerrorstring* = The y-axis extended error information or 0 (zero) if there is no data. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. Use this command if using the SendMotionCommand function. To get any motion errors when running jobs with built in motion, use *GetLastMotionError*.
Also see:	*GetMotionDeviceNames, SendMotionCommand*

GetMotionFileList	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Gets a comma-delimited list of all motion configuration files, and their display names, stored in Flash memory.
Implementation:	"**46**"
Parameters	None
Returns:	*0,motionfile1,motionname1,motionfileN,motionnameN*, a comma-delimited list where: *motionfile* = The name of motion configuration stored in Flash. *motionname* = The name of motion configuration as it appears in WinLase LAN. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Flash memory is the LEC internal storage memory.
Also see:	*LoadMotionFile*

GetMotionStatus	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Gets the x-axis and y-axis status for the specified motion device.
Implementation:	"**65**,devicename"
Parameters	devicename: The name of the motion device.
Returns:	*0,xposition,yposition,xmoving,ymoving,xinputstate,yinputstate*, a comma-delimited list where: *xposition* = The current x-axis position counter value, in steps. *yposition* = The current y-axis position counter value, in steps. *xmoving* = The current x-axis moving state. 0 = stopped, 1 = moving. *ymoving* = The current y-axis moving state. 0 = stopped, 1 = moving. *xinputstate* = The current x-axis Inputs state. *yinputstate* = The current y-axis Inputs state. or *errorcode* if there was an error. *errorcode* is an API response code. The *inputstate* value is a bitmap, where Bit1 represents the input level on the axis Input 1, Bit 2 represents the input level; on the axis Input 2 and so on.
Comments:	Client must call *TakeHostControl* before making this call. Typically, the home and limit switches are connected to the axis Inputs.
Also see:	*GetMotionDeviceNames*

GetNetworkJobFileList	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Gets a comma-delimited list of all jobs stored in a network storage location.
Implementation:	"**221**,subfolder"
Parameters	subfolder
Returns:	*0,joblist* if there was no error, where: *joblist* = A comma-delimited list of all jobs stored in the specified location or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	A connection must already exist to the network share that contains the specified (optional) *subfolder*. Use *ConnectNetworkShare* to connect to a network share. The format of the subfolder entry must be in the format "\subfolder". For example, to specify the Job subfolder under the previously connected network share, use "221,\Job" without quotes. To get a list of files directly in the root of the network share, use "221," without quotes.
Also see:	*ConnectNetworkShare, LoadNetworkJob*

GetNodeFriendlyName	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the local IP address for the LEC.
Implementation:	"**509**"
Parameters	None
Returns:	*name*: The name of the LEC. This name is used in LEC broadcasts.
Comments:	None
Also see:	*SetNodeFriendlyName*

GetObjectCenter	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the geometric center of the specified object in field units.
Implementation:	"**104**,objectindex"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
Returns:	*x*,*y* where: *x* = the x coordinate of the object center *y* = the y coordinate of the object center Units: [bits]
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. The ActiveJob must be *Idle* for this command to succeed. To determine the ActiveJob status, use the *GetJobStatus* command. The *marking field* is described using a Cartesian coordinate system, with: Center: [(0,0)] Bottom Left: [(-fieldsize / 2, -fieldsize / 2)] bits Top Right: [(fieldsize / 2, fieldsize / 2)] bits
Also see:	*GetObjectRect, TransformObject, GetObjectCount, GetObjectType*

GetObjectCount	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the number of objects in the ActiveJob.
Implementation:	"**211**"
Parameters	None
Returns:	*0*,*objectcount* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. There must be an ActiveJob.
Also see:	*GetObjectType*

GetObjectMarkMode	**Supported Platform: Firmware Code 2.x**
Purpose:	Gets the MarkMode for the specified object.
Implementation:	"**120**,objectindex"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
Returns:	*0*,*markmode* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code. *markmode* can be one of the following values: 0 = MarkOnce 1 = MarkMultiple 2 = TwoPassCutClean 3 = ThreePassCutClean 4 = FourPassCutClean
Comments:	None
Also see:	*SetObjectMarkMode, SetObjectNumPasses, GetObjectNumPasses*

GetObjectName	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the name of the specified object.
Implementation:	"**108**,objectindex"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
Returns:	0,objectname if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call.
Also see:	None

GetObjectNumMarkingPasses	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Gets the current number of MarkingPasses for the specified object and the specified vector list.
Implementation:	"**122**,objectindex,vectorlist"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)] vectorlist: The vector list type. 0 = outline list, 1 = fill list. Data type: int
Returns:	*0,numpasses* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	A marking object can have an unlimited number of *MarkingPasses* for both outline vectors and fill vectors. Usually a *MarkingPass* represents a *Profile*. Each *MarkingPass* can also have a number of repetitions.
Also see:	*AddObjectMarkingPass, DeleteObjectMarkingPass, SetObjectPassRepetitions, GetObjectPassRepetitions*

GetObjectNumPasses	**Supported Platform: Firmware Code 2.x**
Purpose:	Gets the current number of passes for the specified object.
Implementation:	"**118**,objectindex"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
Returns:	*0,numpasses* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	The NumPasses property of an object is only valid if the *MarkMode* is *MarkMultiple*.
Also see:	*SetObjectNumPasses, SetObjectMarkMode, GetObjectMarkMode*

GetObjectPassRepetitions	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Get the number of repetitions currently set for the specified *MarkingPass*.
Implementation:	"**126**,objectindex,vectorlist,passindex"
Parameters	objectindex: The zero-based index of the object to change. Data type: int Valid range: [0 to (object count – 1)]
	vectorlist: The vector list profile to set. 0 = outline list, 1 = fill list. Data type: int
	passindex: The index value of the MarkingPass to delete. Data type: int Valid range: [0 to (numpasses – 1)]
Returns:	*0,repetitions* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	The Object at *objectindex* must be a valid marking object type. A marking object can have an unlimited number of *MarkingPasses* for both outline vectors and fill vectors. Usually a *MarkingPass* represents a *Profile*. Each *MarkingPass* can also have a number of repetitions.
Also see:	*SetObjectProfile, GetObjectProfile, GetObjectNumMarkingPasses, AddObjectMarkingPass, DeleteObjectMarkingPass, SetObjectPassRepetitions*

GetObjectProfile	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Get the profile of a marking object contained in the ActiveJob.
Implementation:	"**116**,objectindex,vectorlist,profileindex
Parameters	objectindex: The zero-based index of the object to get the data from. Data type: int Valid range: [0 to (object count – 1)]
	vectorlist: the vector list profile to get. 0 = outline list, 1 = fill list. Data type: int Valid values: Firmware 2.x: [0] Firmware 6.x/7.x [0,1]
	Profile index: Index of profile to get. Data type: int Valid range: [0 to 7]
Returns:	*0,markspeed,jumpspeed,jumpdelay,markdelay,polydelay,laserpower,laseroffdelay,laserondelay,zaxis,frequency,pulsewidth,eightbitword,varijumpdelay,varijumplength,wobbleamplitude,wobblefrequency,powerreset,varipolydelay,powerrampstart,powerrampend,powerrampstartrate,powerrampendrate,powerrampendtime* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. See *SetObjectProfile* for a definition of each returned parameter.
Also see:	*SetObjectProfile, SetObjectProfileFromFile*

GetObjectString	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the string value of the specified text object.
Implementation:	"**107**,objectindex"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
Returns:	*0,objectstring* if there was no error or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. The specified object must be a DynamicText object.
Also see:	*SetObjectString, GetObjectType*

GetObjectType	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the type of the specified object.
Implementation:	"**105**,objectindex"
Parameters	Objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)
Returns:	*0,objecttype* if there was no error (see table below for a list of object types), or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. There must be an ActiveJob.
Also see:	*GetObjectCount*

Object Types:

Value/Constant	Description
0	Polyline
1	Barcode
2	Text
3	Bitmap
4	Vector Graphic
5	Point
6	Line
7	Polygon
8	Rectangle
9	Rounded Rectangle
10	Spiral
107	Laser Control
108	Set Port
109	Time delay
110	Wait Port
111	Alignment
115	Rotary Motion
116	Linear Motion
117	XY Motion

GetObjectRect	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the coordinates of the bounding rectangle for the specified object
Implementation:	"**103**,objectindex"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
Returns:	*left,top,right,bottom* where: *left* = the leftmost bounds of the vectorlist *top* = the topmost bounds of the vectorlist *right* = the rightmost bounds of the vectorlist *bottom* = the bottommost bounds of the vectorlist Units: [bits]
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. The *marking field* is described using a Cartesian coordinate system, with: Center: [(0,0)] Bottom Left: [(-fieldsize / 2, -fieldsize / 2)] bits Top Right: [(fieldsize / 2, fieldsize / 2)] bits
Also see:	*GetObjectCenter, TransformObject, GetObjectCount, GetObjectType*

GetObjectUnicodeString	**Supported Platform: Firmware Code 7.x**
Purpose:	Get the string value in Unicode format, encoded in Base64, of a string based marking object contained in the ActiveJob
Implementation:	"**138**,objectindex,characterindex"
Parameters	objectindex: The zero-based index of the object Valid range: [0 to (object count – 1)] characterindex: The zero-based index of the first character in the string to read. Valid range: [0 to (object count – 1)]
Returns:	*0,charactercount,charactersreturned,base64string* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code. *charactercount* = the total number of characters in the string *charactersreturned* = the number of characters returned in this response, which may be less than the total number of characters. *base64string* = the Base64 formatted string. Decode *base64string* to obtain the actual Unicode string.
Comments:	The Object at *objectindex* must be a valid string based object, such as a text or barcode object. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJobstatus, use the *GetJobStatus* command. *GetObjectUnicodeString* may return less than the total number of characters contained in the actual string. Call *GetObjectUnicodeString* repeatedly and increment *characterindex* to retrieve all the characters in the string.
Also see:	*SetObjectUnicodeString*

GetObjectUserData	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the specified User Data stored by the specified object.
Implementation:	"**110**,objectindex,dataindex"
Parameters	objectindex: The zero-based index of the object Valid range: [0 to (object count – 1)]
	dataindex: The zero-based index of the data. Valid range: [0 and 1]
Returns:	*0,datastring* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	Client must call *TakeHostControl* before making this call. All objects have two string based data buffers (with index 0 and 1) available for the programmer to use for any reason. The maximum size of the buffers is 256. The data buffers are cleared when the object is first loaded, and are *not* persistent between job loads. The object does not use the data contained in the buffers.
Also see:	*SetObjectUserData*

GetObjectVectors	**Supported Platform: Firmware Code 7.x**
Purpose:	Gets an array of floating point values, encoded as a Base64 string, that describe the outline vector list of the specified object.
Implementation:	"**139**,objectindex,entryindex"
Parameters	objectindex: The zero-based index of the object Valid range: [0 to (object count – 1)]
	entryindex: The zero-based index of the list entries Valid range: [0 to (totalentries – 1)]
Returns:	*0,vectorcount,vectorsreturned,base64string* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code. *vectorcount* = the total number of outline vectors *vectorsreturned* = the number of vectors returned in this response, which may be less than the total number of vectors. *base64string* = the Base64 formatted string. Decode *base64string* to obtain the floating point array. The dimension of the float array is *vectorsreturned*.

GetObjectVectors (cont.)

Comments:

The floating point values in the array are arranged in sets of three:
Opcode,Parameter1,Parameter2,Opcode,Parameter1,Parameter2…

Opcode definitions:

Opcode	Description	Parameter1	Parameter2
0	Jump_Abs – move mirrors with laser off	X in µm	Y in µm
1	Laser_On – fire laser for fixed time	Time in ms	0
2	Long_Delay – pause list execution	Time in ms	0
3	Mark_Abs – laser ON, move mirrors, laser OFF	X in µm	Y in µm
4	PolyA_Abs – laser ON, move mirrors	X in µm	Y in µm
5	PolyB_Abs – move mirrors	X in µm	Y in µm
6	PolyC_Abs – move mirrors, laser OFF	X in µm	Y in µm
7	Set_Pen	Pen num	0
13	Begin_Character – start of character outline	0	0
14	End_Character – end of character outline	0	0
15	Text_Radius	Radius	0
16	Text_Ascent	Ascent	0
17	Text_Descent	Descent	0
22	Jump_Fire – move mirrors with laser off (points)	X in µm	Y in µm
23	Begin_Circle – small circle definition		
24	End_Circle – small circle definition		

GetObjectVectors may return less than the total number of vectors contained in the object. Call *GetObjectVectors* repeatedly and increment entryindex to retrieve all the vectors in the object.

Also see: *SetObjectVectors*

GetRemoteIP	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the IP address of the Client that has exclusive control of the LEC.
Implementation:	"**9**"
Parameters	None
Returns:	*remoteIP*: The remote IP address in dot notation format, ex. 192.168.42.1
Comments:	This value is only valid if the Client that has exclusive control is using a TCP/IP connection. All other host interfaces will report "0.0.0.0"
Also see:	None

GetSubnetMask	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the Subnet mask of the LEC
Implementation:	"**511**"
Parameters	None
Returns:	*subnetmask*: The Subnet mask in dot notation format, ex. 255.255.255.0
Comments:	None
Also see:	*SetSubnetMask*

GetUSBJobFileList	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets a comma-delimited list of all jobs stored in a USB drive.
Implementation:	"**204**"
Parameters	None
Returns:	*0,joblist* if there was no error, where: *joblist* = A comma-delimited list of all jobs stored in the USB drive. or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	If no USB drive is found, *NoDrive* is returned. Please check with the factory for tested and approved USB devices.
Also see:	*GetFlashJobFileList*

GetUserInWord	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current User In input port as a 20 bit WORD.
Implementation:	"**30**"
Parameters	None
Returns:	*WORD:* A 20-bit WORD representing the state of the User In 1 – User In 20 inputs.
Comments:	The LSB of the WORD indicates User In 1. A bit value of 0 indicates the input is LOW, a value of 1 indicates the input is HIGH.
Also see:	*SetUserOutBit, GetAllIOWords, GetUserOutInitWord*

GetUserOutInitWord	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the User Out WORD used to set the User Out outputs at system start up and after an Abort.
Implementation:	"**33**"
Parameters	None
Returns:	*WORD:* A 20-bit WORD representing the state the User Out ports will be set to at start up, after an Abort and after an Interlock event.
Comments:	The LSB of the WORD indicates User Out 1. A bit value of 0 indicates the output will be set LOW, a value of 1 indicates the output will be set HIGH.
Also see:	*SetUserOutBit, GetAllIOWords, GetUserInWord*

GetUserOutPreferences	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the UserOut preference flags.
Implementation:	"**41**"
Parameters	None
Returns:	*0,alwaysresetoutputs,setrequireshostcontrol* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
	alwaysresetoutputs controls whether the User Out 1 – 20 outputs are set to the *UserOutInit* value on an Abort or Interlock condition. 0 = Output states are not changed on Abort or Interlock 1 = Output states are set to *UserOutInit* on Abort or Interlock (Default)
	setrequireshostcontrol controls whether the User Out 1 – 20 outputs can be set by the Remote API if the Host has not taken Host Control. 0 = Host can set outputs without taking Host Control. 1 = Host must take Host Control before setting outputs (Default)
Comments:	None
Also see:	*SetUserOutPreferences, SetUserOutBit, SetUserOutWord*

GetUserPIN	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the current User PIN
Implementation:	"**513**"
Parameters	None
Returns:	*userpin*: A string representing the UserPIN
Comments:	The *UserPIN* is used with the Pendant interface, and provides password protection for User access functions.
Also see:	*SetUserPIN, GetAdminPIN, SetAdminPIN*

GetVersions	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Gets the version information for the FPGA firmware, the operating system version and the application executable version.
Implementation:	"**43**"
Parameters	None
Returns:	*fpgaversion,os version,application version*
Comments:	None
Also see:	None

GoToXYZ	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Commands the x, y and z-axes to jump to the specified coordinate, at the specified jump speed, and inserts the specified jump delay after the jump.
Implementation:	"**20**,xcoordinate,ycoordinate,zcoordinate,jumpspeed,jumpdelay"
Parameters	xcoordinate: The coordinate location, to jump to. Valid range: [-fieldsize / 2 to fieldsize / 2] bits
	ycoordinate: The coordinate location, to jump to. Valid range: [-fieldsize / 2 to fieldsize / 2] bits
	zcoordinate: The coordinate location, in bits, to jump to. Valid range: [-32768 to 32767]
	jumpspeed: The speed, in bits/millisecond, that the jump is executed at. Valid range: [1 to 65535]
	jumpdelay: The speed, in microseconds, to delay after the jump is executed. Valid range: [1 to 65535]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. *GoToXYZ* is a *Control Command*, and the instructions are placed in the FIFO immediately. The marking field is described using a Cartesian coordinate system, with: Center: [(0,0)] Bottom Left: [(-fieldsize / 2, -fieldsize / 2) bits Top Right: [(fieldsize / 2, fieldsize / 2) bits
Also see:	*GoToZ*

GoToZ	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Commands the z-axis to jump to the specified coordinate, and inserts the specified jump delay after the jump.
Implementation:	"**19**,zcoordinate,jumpdelay"
Parameters	zcoordinate: The coordinate location, in bits, to jump to. Valid range: [-32768 to 32767]
	jumpdelay: The time, in microseconds, to delay after the jump is executed. Valid range: [1 to 65535]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. *GoToZ* is a *Control Command*, and the instructions are placed in the FIFO immediately.
Also see:	*GoToXYZ*

Hardware Reset	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Resets the LEC
Implementation:	"**8**"
Parameters	None
Returns:	An API Response code
Comments:	After receiving this command, the LEC will perform a soft reset. Any changes made to the IP address parameters will be applied at this time. If you are using a TCP/IP based interface, the socket connection will be closed before the soft reset, and you will have to reconnect after the reset.
Also see:	None

LoadFlashJob	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Loads a job from flash memory into RAM, and sets the job as the ActiveJob.
Implementation:	"**205**,jobname.dat"
Parameters	*jobname*: The job file name in flash to load. Use the filename with the extension, for example, "circle.dat" or "rectangle.job". There can be multiple jobs loaded in RAM simultaneously.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Before interacting with a job, it must be made active with the *MakeJobActive* command.
Also see:	*GetFlashJobList, ExecuteJobContinuous, ExecuteJobOnce, MakeJobActive*

LoadHardwareDefaults	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Loads the currently configured laser, lens and controller parameters from Flash.
Implementation:	"**7**"
Parameters	None
Returns:	An API Response code
Comments:	The *laser, lens* and *controller* parameters are stored in permanent memory on the LEC. This command will reload the parameter sets from the configuration files in permanent memory and may take up to 20 seconds to complete.
Also see:	None

LoadLaserFile	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Loads a laser driver file from Flash and sets it as the active laser configuration.
Implementation:	"**45**,laserfile.xml"
Parameters	*laserfile*: The laser driver file in flash to load. Use the filename with the extension, for example, "laser100.xml".
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command.
Also see:	*GetLaserFileList, TakeHostControl, GetJobStatus*

LoadLensFile	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Loads a lens correction table from Flash and sets it as the active table.
Implementation:	"**39**,lensfile.xml"
Parameters	*lensfile*: The lens correction table in flash to load. Use the filename with the extension, for example, "1001010.xml"
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command.
Also see:	*GetLensFileList, TakeHostControl, GetJobStatus*

LoadMotionFile	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Not implemented
Implementation:	"**47**,motionfile.xml"
Parameters	*motionfile*: motion configuration file in flash to load. Use the filename with the extension, for example, "motioncfg.xml"
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command.
Also see:	*GetMotionFileList, TakeHostControl, GetJobStatus*

LoadNetworkJob	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Loads a job from a network location into RAM, and sets the job as the ActiveJob.
Implementation:	"**222**,jobpath"
Parameters	jobpath: Relative path of the job file to load. Use the filename with the extension, for example, "subfolderpath\circle.dat". There can be multiple jobs loaded into RAM simultaneously.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Before interacting with a job, it must be made active with the *MakeJobActive* command. The *jobpath* can contain subfolder locations relative to the connected network share. For example, to load a job that is in the path\jobs\current\circle.dat under the connected network share server01, use "222,\jobs\current\circle.dat" without quotes. This will load the job \\server01\jobs\current\circle.dat.
Also see:	*GetNetworkJobFileList, ExecuteJobContinuous, ExecuteJobOnce, MakeJobActive*

Load USBJob	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Loads a job from the external USB drive into RAM, and sets the job as the ActiveJob.
Implementation:	"**206**,jobname.dat"
Parameters	jobname: The job file name on the USB drive to load. Use the filename with the extension, ex. "circle.dat". There can be multiple jobs loaded into RAM simultaneously. The job must be located on the USB drive at the following path: \LEC\Jobs.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Before interacting with a job, it must be made active with the *MakeJobActive* command.
Also see:	*GetUSBJobList, ExecuteJobContinuous, ExecuteJobOnce, MakeJobActive*

MakeJobActive	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets a job currently loaded into RAM as the ActiveJob.
Implementation:	"**201**,jobname"
Parameters	jobname: The job file name to make active. Use the filename with the extension, ex. "circle.dat"
Returns:	An API Response code
Comments:	Before interacting with a job, it must be made active with the *MakeJobActive* command.
Also see:	*LoadFlashJob, LoadUSBJob, GetActiveJob*

NewObject	**Supported Platform: Firmware Code 7.x**
Purpose:	Creates a new object with the specified properties and adds it to the ActiveJob.
Implementation:	"**136**,objectID,objectname,properties"
Parameters:	objectID: The type of object to create. See table below for a list of object types.
	objectname: the name of the object.
	properties: Optional. Additional properties that depend on the object type (future use).

Object Types:

Value/Constant	Description
0	Polyline
1	Barcode
2	Text
3	Bitmap
4	Vector Graphic
5	Point
6	Line
7	Polygon
8	Rectangle
9	Rounded Rectangle
10	Spiral
107	Laser Control
108	Set Port
109	Time delay
110	Wait Port
111	Alignment
115	Rotary Motion
116	Linear Motion
117	XY Motion

NewObject (cont.)	
Returns:	*0,objectindex,objectname* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code. *objectindex* = the index of the new object in the job. *objectname* = the actual name assigned to the object. If another object in the job has the same name as the one specified, the name will be appended to create a unique name.
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed.
Also see:	*DeleteObject, NewJob*

NewJob	**Supported Platform: Firmware Code 7.x**
Purpose:	Create a new empty job in memory and returns the name of the new job.
Implementation:	"**225**"
Parameters	None
Returns:	*0,jobname* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code. *jobname* = the name that was assigned to the job when it was created.
Comments:	Client must call *TakeHostControl* before making this call.
Also see:	*RemoveJob, MakeJobActive*

OpenCOMPort	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Opens the specified COM port on the LEC at the specified baud rate.
Implementation:	"**16**,portnum,baudrate"
Parameters	portnum: The COM port to open Valid range: [1,2,3]
	baudrate: The port baud rate Valid range: [110,300,1200,2400,4800,9600,19200,38400,57600,115200]
Returns:	An API Response code
Comments:	The COM port is opened with the specified baud rate, 8 data bits, 1 stop bit, no parity, and no flow control.
Also see:	*CloseCOMPortEx*

ReleaseHostControl	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Release the exclusive control of the LEC back to the LANStream host.
Implementation:	"**3**"
Parameters	None
Returns:	An API Response code
Comments:	When any Client releases exclusive control, the *LANStream* host is given exclusive control. The *LANStream* host is the streaming interface used when connected to the device with WinLase LAN.
Also see:	*TakeHostControl*

RemoveJob	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Deletes the ActiveJob from memory
Implementation:	"**202**"
Parameters	None
Returns:	An API Response code
Comments:	The ActiveJob is cleared after this call completes. To set the Active job, call *MakeJobActive*
Also see:	*NewJob, LoadFlashJob, LoadUSBJob, MakeJobActive*

RemoveObject	**Supported Platform: Firmware Code 7.x**
Purpose:	Deletes the object at *objectindex* from the ActiveJob
Implementation:	"**141**,objectindex"
Parameters	None
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed.
Also see:	*NewObject*

ResetObject	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Restores the Object outline and fill vector lists to the state they were in when the object first loaded from the job.
Implementation:	"**111**,objectindex"
Parameters	objectindex: The zero-based index of the object to change. Valid range: [0 to (object count – 1)]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This call resets the current Object outline and fill vector lists to the lists that were created when the job first loaded. This call does not reset the (optional) current string serialization value. To reset the current value, the job must be reloaded.
Also see:	*TakeHostControl, GetJobStatus, TransformObject, SetObjectString*

ResetPerformanceGlobals	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Resets the Performance global values to their defaults
Implementation:	"**15**"
Parameters	None
Returns:	An API Response code
Comments:	All marking objects have a global performance data structure applied before marking. Calling *ResetPerformanceGlobals* resets the performance data structure to default values.
Also see:	*SetPerformanceGlobals*

ResetUserTransform	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Resets the *UserTransform* to the Identity matrix.
Implementation:	"**112**,objectindex"
Parameters	objectindex: The zero-based index of the object to change. Valid range: [0 to (object count – 1)]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. This call only resets the transform matrix for the object. To reset the object's vectors, use *ResetObject* after calling *ResetUserTransform*.
Also see:	*TransformObject, ResetObject*

SampleMOTFEncoderCount	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Clears the MOTF encoder counter to zero, waits for the sample period, then reads the encoder count.
Implementation:	"**34**,sampleperiod"
Parameters	sampleperiod: The time, in ms, to wait after clearing the counter to zero before reading the counter value.
Returns:	*0,count* if there was no error, or *errorcode* if there was an error. *errorcode* is an API response code.
Comments:	The value count is in units of field bits. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. The LEC must have a valid Advanced (MOTF) license.
Also see:	*ClearMOTFEncoderCount, GetMOTFEncoderCount, GetJobStatus*

SaveFlashJob	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Save the ActiveJob to Flash memory.
Implementation:	"**219**,jobname.dat"
Parameters	jobname: The file name to assign to the job when saved. If this parameter is empty (i.e. "219" with no comma), the ActiveJob will be saved to flash with the file name that was used to load the job. Use the filename with the extension, for example, "circle.dat".
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Before interacting with a job, it must be made active with the *MakeJobActive* command.
Also see:	*NewJob, LoadFlashJob, GetFlashJobList*

SaveNetworkJob	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Save the ActiveJob to a network location.
Implementation:	"**223**,jobpath"
Parameters	jobpath: The path to the network location to save the job, and the file name to assign to the job when saved. Use the filename with the extension, for example, "\Job\circle.job".
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Before interacting with a job, it must be made active with the *MakeJobActive* command. The *jobpath* can contain subfolder locations relative to the connected network share. For example, to save a job to the path \jobs\current\circle.dat under the connected network share server01, use "223,\jobs\current\circle.job" without quotes. This will save the ActiveJob to the location \\server01\jobs\current.
Also see:	*LoadNetworkJob, GetNetworkJobList*

SaveUSBJob	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Save the ActiveJob to the external USB drive.
Implementation:	"**220**,jobname.dat"
Parameters	jobname: The file name to assign to the job when saved. If this parameter is empty (i.e. "220" with no comma), the ActiveJob will be saved to the USB drive with the file name that was used to load the job. When specifying the name, use the filename with the extension, for example, "circle.dat". The job file will be saved to the USB drive path: \LEC\Jobs.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. Before interacting with a job, it must be made active with the *MakeJobActive* command.
Also see:	*LoadUSBJob, GetUSBJobList*

SendMotionCommand	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Sends a motion command to the specified device.
Implementation:	"**61**,devicename,movemethod,axis,x,y,initialvelocity,slewvelocity,accelrate,decelrate,movedelay
Parameters	devicename: The name of the motion device.
	movemethod: The type of move. Valid values: [0,1,2,3,4,5,6,7,8] Disabled = 0 Absolute = 1 Relative = 2 Home = 3 Stop = 4 Continuous (X+)(Y+) = 5 Continuous (X-)(Y-) = 6 Continuous (X+)(Y-) = 7 Continuous (X-)(Y+) = 8

SendMotionCommand (cont.)	
Parameters	axis: The specific axis to send the command to. For a linear or rotary device, only the x-axis is supported. For an XY device, both the x-axis and y-axis is supported. Valid values: [0,1,2,3] Disabled = 0 X-Axis = 1 Y-Axis = 2 X + Y Axis = 3
	x: The x-axis move command. Linear and XY: in µm, Rotary: in µdegrees
	y: (XY devices only) The y-axis move command, in µm.
	initialvelocity: The velocity the axis uses when it first starts the move. Linear and XY: in µm/sec, Rotary: in µdegrees/sec.
	slewvelocity: The velocity the axis uses after accelerating. Linear and XY: in µm/sec, Rotary: in µdegrees/sec.
	accelrate: The rate of acceleration at the start of the move. Linear and XY: in µm/sec^2, Rotary: in µdegrees/sec^2.
	decelrate: The rate of deceleration at the end of the move. Linear and XY: in µm/sec^2, Rotary: in µdegrees/sec^2.
	movedelay: Time to wait, in ms, after the move is complete before setting the move flag to false.
Returns:	AN API Response code
Comments:	Client must call *TakeHostControl* before making this call. If the device is currently executing a move, this call will fail.
Also see:	*GetMotionDeviceNames, GetMotionStatus, GetMotionErrorCodes*

SetActiveLocalDevice	**Supported Platforms: WinXP (SP3), Win7 (Pro), Win8/8.1 (Pro)**
Purpose:	Creates a connection to an LEC device specified by the deviceid. All communication will be directed to this device.
Implementation:	"**527**,deviceid"
Parameters	deviceid: A string representing a detected LEC device.
Returns:	An API Response code
Comments:	Devices can be discovered by using the *GetLocalDeviceList* command.
Also see:	*GetLocalDeviceList*

SetAdminPin	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the AdminPIN
Implementation:	"**500**,pin"
Parameters	pin: The numeric based Administration password
Returns:	An API Response code
Comments:	The *AdminPIN* is used with the Pendant interface, and provides password protection for Administrative functions.
Also see:	*GetAdminPIN, SetUserPIN, GetUserPIN*

SetCOMPortAssignments	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the COM port to be used for the Pendant, RS-232 Remote API and the Motion interfaces. On Platforms that use the 6.x and 7.x firmware code, this API has been deprecated and users should use *SetCOMPortMode* instead.
Implementation:	"**516**,pendant,remoteapi,motioncontrol"
Parameters	pendant: The COM port to use for the Pendant interface. Valid range: [0,1,2,3]
	remoteapi: The COM port to use for the Remote API interface. Valid range: [0,1,2,3]
	motioncontrol: The COM port to use for the Motion Control interface. Valid range: [0,1,2,3]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued. There are three COM ports available on the LEC: COM1, COM2, COM3. When issuing this command, you must assign a different COM port to each interface, or the command will fail. Use a 0 (zero) to clear a COM port assignment from the interface.
Also see:	*GetCOMPortAssignments, SetCOMPortSpeed, GetCOMPortSpeed*

SetCOMPortMode	**Supported Platforms: Firmware Code 6.x and 7.x (starting with firmware versions 6.0.2.2 and 7.0.2.2 respectively)**
Purpose:	Sets the interface mode for three available COM ports.
Implementation:	"**528**,com1mode,com2mode,com3mode"
Parameters	com1mode: The COM1 interface mode. Valid range: [0,1,2,3]
	com2mode: The COM2 interface mode. Valid range: [0,1,2,3]
	com3mode: The COM3 interface mode. Valid range: [0,1,2,3]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The LEC device must be restarted for changes to take effect. There are three COM ports available on the LEC: COM1, COM2 and COM3. The interface modes are defined as: 0 = unassigned 1 = Motion interface 2 = Remote API interface 3 = Pendant interface The Motion and/or Pendant interface can only be assigned to a single COM port or a *BadArg* response will be returned. The "unassigned" interface and the Remote API interface may be assigned to multiple COM ports.
Also see:	*GetCOMPortMode, SetCOMPortSpeedEx, GetCOMPortSpeedEx*

SetCOMPortSpeed	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (up to firmware versions 2.2.27.4, 6.0.1.36 and 7.0.1.36 respectively)**
Purpose:	Sets the baud rate of the specified interfaces that use the COM ports. On Firmware Code versions 6.x and 7.x, this API has been deprecated and users should use *SetCOMPortSpeedEx* instead.
Implementation:	"**514**,pendantspeed,apispeed,motionspeed"
Parameters	pendantspeed: The desired baud rate of the COM port that the pendant is assigned to. Valid values: [110,300,600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200]
	apispeed: The desired baud rate of the COM port that the Remote API is assigned to. Valid values: [110,300,600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200]
	motionspeed: The desired baud rate of the COM port that the Motion controller is assigned to. Valid values: [110,300,600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued. There are three COM ports available on the LEC: COM1, COM2, COM3. Only the baud rate can be configured on the ports. The remaining settings are: 8 data bits No parity 1 stop bit XON/XOFF (Software handshake)
Also see:	*GetCOMPortSpeed, SetCOMPortAssignments, GetCOMPortAssignments*

SetCOMPortSpeedEx	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x (from firmware versions 2.2.27.5, 6.0.1.37 and 7.0.1.37 respectively)**
Purpose:	Set the baud rates on the COM ports.
Implementation:	"**524**,com1speed,com2speed,com3speed"
Parameters	com1speed: The desired baud rate of the COM1 port. Valid values: [110,300,600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200]
	com2speed: The desired baud rate of the COM2 port. Valid values: [110,300,600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200]
	com3speed: The desired baud rate of the COM3 port. Valid values: [110,300,600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued. There are three COM ports available on the LEC: COM1, COM2, COM3. Only the baud rate can be configured on the ports. The remaining settings are: 8 data bits No parity 1 stop bit No hardware control
Also see:	*GetCOMPortSpeedEx, SetCOMPortMode, GetCOMPortMode*

SetDHCPMode	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the current DHCP mode.
Implementation:	"**502**,mode"
Parameters	Mode: "StaticIP" or "Autodetect"
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued. If the LEC is in DHCP mode, at power up it will attempt to get an IP address from a DHCP server on the local network.
Also see:	*TakeHostControl, HardwareReset, GetDHCPMode*

SetExternalStartMode	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the current ExternalStart mode of the ActiveJob.
Implementation:	"**215**,mode"
Parameters	mode: 0 = When port is HIGH 1 = When port is LOW 2 = After transition from LOW > HIGH 3 = After transition from HIGH > LOW
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. The External Start mode controls what type of signal transition on the Start Mark input will trigger the start of job execution.
Also see:	*GetExternalStartMode*

SetLocalGateway	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the default local Gateway address for the LEC.
Implementation:	"**504**,gateway"
Parameters	gateway: The Gateway address in dot notation format, ex. 192.168.42.1
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued.
Also see:	*TakeHostControl, HardwareReset, GetLocalGateway*

SetLocalIP	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the IP address of the LEC
Implementation:	"**506**,ipaddress"
Parameters	ipaddress: The local IP address, in dot notation format, ex. 192.168.42.1
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued.
Also see:	*TakeHostControl, HardwareReset*

SetLocalTime	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the current local time and date.
Implementation:	"**518**,year,month,day,hour,minute,second"
Parameters	year: Specifies the current year
	month: Specifies the current month; January = 1, February = 2 and so on.
	day: Specifies the current day of the month.
	hour: Specifies the current hour in 24 hour time format.
	minute: Specifies the current minute.
	second: Specifies the current second.
Returns:	An API Response code
Comments:	None
Also see:	*GetLocalTime*

SetMemBuffer	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the internal memory buffer at the specified index.
Implementation:	"**22**,bufferindex,newstring"
Parameters	bufferindex: The index of the memory buffer to set. Valid range: [1 to 10]
	newstring: The string to save in the specified memory buffer. Valid range: [1 to 2999 characters]
Returns:	An API Response code
Comments:	A string based object (barcodes and text) can be configured to retrieve the next string to mark from an internal memory buffer at execution time. There are 10 buffers available.
Also see:	*GetMemBuffer*

SetMOTFEncoderRate	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the current MOTF encoder rate in the ActiveJob, which will take effect the next time a job is executed.
Implementation:	"**21**,rate"
Parameters	rate: The encoder rate, in bits/count. When the LEC is configured for simulated encoder mode, a fixed encoder pulse rate of 1MHz is used, so the encoder rate is the distance the part moves (in field bits) through the field during 1 clock tick (1/1,000,000 second). When using an actual encoder, the encoder rate equals the distance the part moves (in field bits) through the field during one encoder pulse. Example with simulated encoder mode: Line Speed = 100 mm/s KFactor = 570 bits/mm Encoder pulse rate = 1,000,000 counts/s [(Line speed) x (KFactor)] / 1,000,000 = 0.057 bits/count Valid range: [-32768.000 to 32767.000] Default: [0] (no MOTF)

SetMOTFEncoderRate (cont.)	
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call, and the ActiveJob must be a job that has MOTF enabled. The LEC must have a valid Advanced (MOTF) license.
Also see:	*TakeHostControl, MakeJobActive, GetKFactor*

SetNodeFriendlyName	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the name of the LEC
Implementation:	"**508**,name"
Parameters	name: The name of the LEC. This name is used in LEC broadcasts.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call.
Also see:	*TakeHostControl*

SetObjectMarkMode	**Supported Platform: Firmware Code 2.x**
Purpose:	Set the MarkMode property on the specified object.
Implementation:	"**121**,objectindex,markmode"
Parameters	objectindex: The zero-based index of the object to change. Valid range: [0 to (object count – 1)]
	markmode: MarkMode property. Valid values: [0 to 4], where 0 = MarkOnce 1 = MarkMultiple 2 = TwoPassCutClean 3 = ThreePassCutClean 4 = FourPassCutClean
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. If MarkMultiple is specified, use *SetObjectNumPasses* to control the number of passes.
Also see:	*GetObjectNumPasses, SetObjectNumPasses, GetObjectMarkMode*

SetObjectNumPasses	**Supported Platform: Firmware Code 2.x**
Purpose:	Set the NumPasses property on the specified object.
Implementation:	"**119**,objectindex,numpasses"
Parameters	objectindex: The zero-based index of the object to change. Valid range: [0 to (object count – 1)]
	numpasses: NumPasses property. Valid size: [> 1]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call.
Also see:	*GetObjectNumPasses, SetObjectMarkMode, GetObjectMarkMode*

SetObjectPassRepetitions	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Set the number of repetitions for the specified *MarkingPass*.
Implementation:	**125**,objectindex,vectorlist,passindex,repetitions"
Parameters	objectindex: The zero-based index of the object to change. Data type: int Valid range: [0 to (object count – 1)]
	vectorlist: The vector list profile to set. 0 = outline list, 1 = fill list. Data type: int
	passindex: The index value of the MarkingPass to delete. Data type: int Valid range: [0 to (numpasses – 1)]
	repetitions: The number of times to mark the *MarkingPass.* Data type: int Valid range: [> 0]
Returns:	An API Response code
Comments:	The Object at *objectindex* must be a valid marking object type. A marking object can have an unlimited number of *MarkingPasses* for both outline vectors and fill vectors. Usually a *MarkingPass* represents a *Profile*. Each *MarkingPass* can also have a number of repetitions.
Also see:	*SetObjectProfile, GetObjectProfile, GetObjectNumMarkingPasses, AddObjectMarkingPass, DeleteObjectMarkingPass, GetObjectPassRepetitions*

SetObjectProfile	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the profile of a marking object contained in the ActiveJob.
Implementation:	"**115**,objectindex,vectorlist,profileindex,markspeed,jumpspeed,jumpdelay,markdelay,polydelay,laserpower,laseroffdelay,laserondelay,zaxis,frequency,pulsewidth,eightbitword,varijumpdelay,varijumplength,wobbleamplitude,wobblefrequency,powerreset,varipolydelay,powerrampstart,powerrampend,powerampstartrate,powerampendrate,powerrampendtime
Parameters	objectindex: The zero-based index of the object to change. Data type: int Valid range: [0 to (object count – 1)]
	vectorlist: The vector list profile to set. 0 = outline list, 1 = fill list. Data type: int Valid values: Firmware 2.x, 6.x [0] Firmware 7.x [0,1]
	profileindex: Index of profile to change. Data type: int Validrange: [0 to 7]
	markspeed: Defines the speed of the laser spot while marking. Data type: double Valid range: [0.1 to 100000] bits/ms
	jumpspeed: Defines the speed at which the mirrors jump to the next marking vector. Data type: double Valid range: [0.1 to 100000] bits/ms
	jumpdelay: Defines the delay after a jump and before the next marking vector starts. Data type: int Valid range: [0 to 65500] µs

SetObjectProfile (cont.)	
Parameters (cont.)	markdelay: Defines the delay between a marking vector and a jump vector. Data type: int Valid range: [0 to 65500] µs
	polydelay: Defines the delay between contiguous marking vectors. Data type: int Valid range: [0 to 65500] µs
	laserpower: Controls the analog and 8-bit digital laser power outputs on the LEC. Application of this parameter depends on a setting in the laser driver file. See *eightbitword* below. Data type: int Valid range: [0 to 255]
	laseroffdelay: Defines the delay after the last marking vector finishes and the laser is turned off. Data type: int Valid range: [0 to 65500] µs
	laserondelay: Defines the delay after a marking vector starts and the laser is turned on. Data type: int Valid range: [-32000 to 32000] µs
	zaxis: Defines the z-axis defocus value Data type: int Valid range: Firmware 2.x, 6.x [-32768 to 32767] bits Firmware 7.x [-524288 to 524287] bits
	frequency: Defines the frequency of the laser modulation signal output on LaserMod1 and LaserMod2. Data type: double Valid range: [0.02 to 2000.0] kHz
	pulsewidth: Defines the pulse width of the laser modulation signal output on LaserMod1 and LaserMod2. Data type: double Valid range: [0.1 to 65535.0] µs
	eightbitword: An 8-bit value that can be applied to the 8-bit digital laser power output, or the analog output. The port used depends on a setting in the laser driver file. See *laserpower*. Data type: int Valid Range: [0 to 255]
	varijumpdelay: Defines the delay after a jump and before the next marking vector starts if variable jump delay is in effect. Set to 0 (zero) to disable variable jump delay. Data type: int Valid range: [0 to 30000] µs
	varijumplength: Defines the length of a vector, at which any vector that is longer will use the *varijumpdelay* parameter, and any vector that is shorter will use the *jumpdelay* parameter. Data type: int Valid range: Firmware 2.x, 6.x [0 to 65535] bits Firmware 7.x [0 to 1048576] bits
	wobbleamplitude: The diameter of the circle created when the spot is dithered. Set to 0 (zero) to disable Data type: int Valid Range: [0 to 5000] bits

SetObjectProfile (cont.)	
Parameters (cont.)	wobblefrequency: The frequency of the laser spot as it dithers around the circle defined in *wobbleamplitude*. Data type: double Valid range: [0 to 6000] Hz
	powerreset: 0 = do not set laser power to minimum when mark complete, 1 = set laser power to minimum when mark complete Data type: int Valid range: [0 to 1]
	varipolydelay: Reserved. Set to 0
	powerrampstart: Reserved. Set to 0
	powerrampend: Reserved. Set to 0
	powerrampstartrate: Reserved. Set to 0
	powerampendrate: Reserved. Set to 0
	powerrampendtime: Reserved. Set to 0
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type.
Also see:	*SetObjectProfileFromFile, GetObjectProfile*

SetObjectProfileFromFile	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the profile of a marking object in the ActiveJob using the specified profile file at the specified location.
Implementation:	"**117**,objectindex,vectorlist,profileindex,location,filename"
Parameters	objectindex: The zero-based index of the object to change. Data type: int Valid range: [0 to (objectcount – 1)]
	vectorlist: The vector list profile to set. 0 = outline list, 1 = fill list. Data type: int Valid values: Firmware 2.x, 6.x [0] Firmware 7.x [0,1]
	profileindex: Index of profile to change. Data type: int Valid range: [0 to 7]
	location: The location to search for the named file. Valid values: [0 or 1], where 15 = Internal Flash memory 16 = USB drive memory 17 = Network location
	filename: Name of the profile file to load, with file extension. When specifying a network location, subfolders can be specified, ex. "subfolderpath\default.pro"
Returns:	An API Response code

SetObjectProfileFromFile (cont.)	
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. When specifying a network location, the LEC must be connected to a network share by using *ConnectNetworkShare*. The *filename* can contain subfolder locations relative to the connected network share. For example, to load a profile that is in the path: \profiles\current\anodize.pro under the connected network share server01, use "117,\profiles\current\anodize.pro" without quotes. This will load the profile \\server01\profiles\current\anodize.pro.
Also see:	*SetObjectProfile, GetObjectProfile, ConnectNetworkShare*

SetObjectString	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the string value of a string based marking object contained in the ActiveJob
Implementation:	"**100**,objectindex,newstring"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
	newstring: The new string value. Valid size: [1 to 2999 characters] Special conditions: *DataMatrix*: To imbed control characters in the string, use the tilde (~) character before the control code. To imbed an actual ~ character, use two tilde characters in a row (~~). Exception: To imbed an ASCII 0 character (NUL) , use ~@ instead of the ASCII 0.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid string based object, such as a text or barcode object. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command.
Also see:	*GetObjectString*

SetObjectUnicodeString	**Supported Platform: Firmware Code 7.x**
Purpose:	Set the string value of a string based marking object contained in the ActiveJob with a Unicode string formatted in Base 64.
Implementation:	"**137**,objectindex,characterindex,base64string"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
	characterindex: The zero-based character index of the string. Valid range: [0 to (object count – 1)]
	base64string: The new string value formatted as a Basic64 string. Valid size: [1 to 2999 characters]
Returns:	An API Response code

SetObjectUnicodeString (cont.)	
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid string based object, such as a text or barcode object. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. The Remote API uses an ASCII based communication protocol, and so it is not possible to directly send Unicode characters to/from the API. However, converting a Unicode string into a Base64 string allows setting strings with Unicode content. An object can contain a maximum of 2999 Unicode characters. When encoding Unicode strings into a Base64 string, the Base64 string will be approximately 33% larger than the original Unicode string. Because the *base64string* parameter is limited to 2999 characters, *SetObjectUnicodeString* may need to be called multiple times to build a complete string containing the maximum number of Unicode characters. Use the *characterindex* parameter to specify the position in the Objects current string to add the new string. *characterindex* can be any value up to the size of the current string. The new string will be written at the *characterindex* position and will overwrite characters that may already exist.
Also see:	*GetObjectUnicodeString*

SetObjectUserData	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the specified User Data stored by the specified object.
Implementation:	"**109**,objectindex,dataindex,datastring"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
	dataindex: The zero-based index of the data Valid range: [0 and 1]
	datastring: The string to store in the specified data buffer. Valid size: [1 to 255 characters]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. All objects have two string based data buffers (with index 0 and 1) available for the programmer to use for any reason. The maximum size of the buffers is 256. The data buffers are cleared when the object is first loaded, and are *not* persistent between job loads. The object does not use the data contained in the buffers in any way.
Also see:	*GetObjectUserData*

SetObjectVectors	**Supported Platform: Firmware Code 7.x**
Purpose:	Sets the outline vector list in the specified object using a Base64 formatted data string.
Implementation:	"**140**,objectindex,entryindex,base64datastring"
Parameters	objectindex: The zero-based index of the object. Valid range: [0 to (object count – 1)]
	listindex: The zero-based index of the vector list to write the new data.. Valid range: [0 to vectorcount]
	base64datastring: The new array of floating point values formatted as a Base64 string. Valid size: [1 to 2999 characters]
Returns:	An API Response code

SetObjectVectors (cont.)	
Comments:	Client must call *TakeHostControl* before making this call. The ActiveJob must be *Idle* for this command to succeed. The Object at *objectindex* must be a Polyline object. Because *base64datastring* is limited to 2999 characters, *SetObjectVectors* may need to be called multiple times to build a complete vector list in the Object. Use the *listindex* parameter to specify the position in the Objects current vector list to add the new data. *listindex* can be any value up to the size of the current list. The new data will be written at the *listindex* position and will overwrite data that may already exist. There is no error checking of the actual data contained in the floating point array, so **it is the responsibility of the programmer** to ensure the data represents a valid vector list. Take care when concatenating new vectors onto an existing list to make sure the list remains valid
Also see:	*GetObjectVectors*

SetPerformanceGlobals	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the current Performance global data structure values.
Implementation:	"**14**,markspeed,laserpower,pulsewidth,period,orientation,xoffset,yoffset,zoffset"
Parameters	markspeed: Scale factor for the mark speed. Valid range: [0.500 to 1.500] Default: [1.000]
	laserpower: Scale factor for the laser power. Valid range: [0.800 to 1.200] Default: [1.000]
	pulsewidth: Scale factor for the laser modulation signal pulse width. Valid range: [0.500 to 1.500] Default: [1.000]
	period: Scale factor for the laser modulation signal period. Valid range: [0.500 to 1.500] Default: [1.000]
	orientation: Orientation of the scan head to the marking field. Valid values: [0, 90, 180, 270] Default: [0]
	xoffset: Offset of the x-coordinate of all marking objects, in bits. Valid values: [-32768 to 32767] Default: [0]
	yoffset: Offset of the y-coordinate of all marking objects, in bits. Valid values: [-32768 to 32767] Default: [0]
	zoffset: Offset of the z-coordinate of all marking objects, in bits. Valid values: [-32768 to 32767] Default: [0]
Returns:	An API Response code
Comments:	All marking objects have a global performance "adjustment" applied before marking. Use the *SetPerformanceGlobals* function to set the scalar values that are applied to the specified parameters. Not all parameters need to be set at the same time. To leave the current setting of a specific parameter unchanged, use the string "NOP" for that parameter. For example, to only set the *pulsewidth* scalar to 1.2, use the following command: SetPerformanceGlobals,NOP,NOP,1.2,NOP,NOP,0,0,0. Note that the x, y and z offsets must be provided.
Also see:	*ResetPerformanceGlobals*

SetSubnetMask	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the Subnet mask of the LEC.
Implementation:	"**510**,subnetmask"
Parameters	subnetmask: The Subnet mask in dot notation format, ex. 255.255.255.0
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareReset* command is issued.
Also see:	*TakeHostControl, HardwareReset, SetSubnetMask*

SetUserOutBit	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the specified User Out output
Implementation:	"**29**,bitnum,value"
Parameters	bitnum: The User Out output to set, which ranges from User Out 1 – User Out 20. Valid range: [1 to 20]
	value: The logic level to set the output to. Valid values: [0 or 1]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call unless overridden by the *SetUserPreferences* command. The ActiveJob must be *Idle* for this command to succeed. To determine ActiveJob status, use the *GetJobStatus* command. When *value* is 0, the output will be set LOW, when *value* is 1 the output will be set HIGH. ***Note:*** 8-75 Markers support User In bits 1-4 only, 8-77 Markers support User In bit 1-12 only, where bits 5-12 are defined in Appendix B as "Job Select Bit 1-8" All 8-75 and 8-77 Markers support User Out bits 1-4 only.
Also see:	*TakeHostControl, SetUserOutPreferences, SetUserOutWord, GetJobStatus, GetUserInWord, GetAllIOWords*

SetUserOutInitWord	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the User Out output WORD that will be used at system start up and after an Abort.
Implementation:	"**32**,word"
Parameters	word: The User Out WORD. Valid range: [0 to 1048576]
Returns:	An API Response code
Comments:	***Note:*** 8-75 Markers support User In bits 1-4 only, 8-77 Markers support User In bit 1-12 only, where bits 5-12 are defined in Appendix B as "Job Select Bit 1-8" All 8-75 and 8-77 Markers support User Out bits 1-4 only.
Also see:	*TakeHostControl, GetUserInWord, GetAllIOWords, SetUserOutBit*

SetUserOutWord	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Set the User Out output WORD on User Out 1 – 20.
Implementation:	"**42**,word"
Parameters	word: The User Out WORD. Valid range: [0 to 1048576]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call unless overridden by the *SetUserPreferences* command. The ActiveJob must be *Idle* for this command to succeed.
Also see:	*TakeHostControl, SetUserOutPreferences, GetUserInWord, GetAllIOWords, SetUserOutBit, GetJobStatus*

SetUserOutPreferences	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the UserOut preference flags options for the User Out 1 – 20 ports.
Implementation:	"**40**,alwaysresetoutputs,setrequireshostcontrol"
Parameters	alwaysresetoutputs: Flag which controls whether the User Out 1 – 20 outputs are set to the *UserOutInit* value on an Abort or Interlock condition. 0 = output states are not changed on Abort or Interlock 1 = output states are set to *UserOutInit* on Abort or Interlock (Default)
	setrequireshostcontrol: Flag which controls whether the User Out 1- 20 outputs can be set by the Remote API if the Host has not taken Host Control. 0 = Host can set outputs without taking Host Control 1 = Host must take Host Control before setting outputs (Default)
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The value of the two flags are not persistent between power cycles; when the LEC starts up, both flags will be set to their default values.
Also see:	*GetUserOutPreferences, SetUserOutBit, SetUserOutWord*

SetUserPIN	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Sets the current User PIN
Implementation:	"**512**,userpin"
Parameters	userpin: A string representing the UserPIN.
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. This setting will not take effect until the board is power cycled, or a *HardwareRest* command is issued. The *UserPIN* is used with the Pendant interface, and provides password protection for User access functions.
Also see:	*TakeHostControl, HardwareReset, SetUserPIN, GetAdminPIN, SetAdminPIN*

SetZOffsetRWU	**Supported Platforms: Firmware Code 6.x and 7.x**
Purpose:	Sets the global Z offset
Implementation:	"**76**,zoffset"
Parameters	zoffset: The focus offset, in µm, from the nominal focus position of zero. The offset can be positive or negative, but must be within the range of the z focus optics hardware.
Returns:	An API Response code
Comments:	This function is intended to support AF (adjustable) focus shifter applications. The new focus position will be applied the next time a marking object is executed.
Also see:	None

TakeHostControl	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Request exclusive control of the LEC
Implementation:	"**2**"
Parameters	None
Returns:	An API Response code
Comments:	A Client cannot gain exclusive control of the LEC if it is busy processing a job. Use *GetJobStatus* to determine if there is a job currently being processed.
Also see:	*ReleaseHostControl, GetJobStatus*

TransformObject	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Applies rotation, scaling and offset to the specified object and concatenates the transform to the current *UserTransform*.
Implementation:	"**102**,objectindex,rotation,rotationcenterx,rotationcentery,xscale,yscale,xoffset,yoffset"
Parameters	objectindex: The zero-based index of the object to translate. Valid range: [0 to (object count – 1)]
	rotation: The relative amount to rotate the object, in degrees. A positive rotation value results in a clockwise rotation. Valid range: [-360.0 to 360.0]
	rotationcenterx: The coordinate position representing the center of rotation in the x-axis. This value can be a coordinate position that is outside the normal marking field. Valid range: [-2,147,483,648 to 2,147,483,647] Units: [bits]
	rotationcentery: The coordinate position representing the center of rotation in the y-axis. This value can be a coordinate position that is outside the normal marking field. Valid range: [-2,147,483,648 to 2,147,483,647] Units: [bits]
	xscale: Amount to scale object in the x-axis. Valid range: [> 0]
	yscale: Amount to scale object in the y-axis. Valid range: [> 0]
	xoffset: The amount to move the object in the x-axis. Valid range: [-2,147,483,648 to 2,147,483,647] Units: [bits]
	yoffset: The amount to move the object in the y-axis. Valid range: [-2,147,483,648 to 2,147,483,647] Units: [bits]

TransformObject (cont.)	
Returns:	An API Response Code
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. The transformations are applied in the following order: 1. Rotation 2. Scaling 3. Offset Both the Object *outline* and the Object *fill* are transformed with this call. When calling *SetObjectString*, or if the object will change its string value at run-time because of serialization, AutoDate, etc. the Object is processed in the following order: 1. New Outline vectors are generated 2. The JobTransform is applied (from the Job) 3. The Object is Justified 4. 4. New Fill vectors are generated 5. The *UserTransform* is applied Subsequent calls to *TransformObject* are cumulative, as each transform is concatenated to the current *UserTransform*. To clear all transforms (setting the *UserTransform* to the Identity matrix), call *ResetUserTransform*. In order to maximize performance, the Remote API does not check whether an objects vector list is within the legal marking field. Therefore, **it is the responsibility of the programmer** to insure that after an object has been transformed, it is within the legal marking field boundaries. Undefined results can be expected if an attempt is made to execute an object with a vector list outside the legal marking field. The bounds of an object can be discovered by calling *GetObjectRect*. The *marking field* is described using a Cartesian coordinate system, with: Center: [(0,0)] Bottom Left: [(-fieldsize / 2, -fieldsize / 2)] bits Top Right: [(fieldsize / 2, fieldsize / 2)] bits
Also see:	*GetKFactor, GetJobStatus,TakeHostControl, GetObjectType, GetObjectRect, GetObjectCenter, ResetObjectTransform, TransformObjectByName*

TransformObjectNewFill	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Applies rotation, scaling, and offset to the specified object outline, generates new fill, then concatenates the transform values to the current *UserTransform*.
Implementation:	"**127**,objectindex,rotation,rotationcenterx,rotationcentery,xscale,yscale,xoffset,yoffset"
Parameters	See *TransformObject* for parameter definitions.
Returns:	An API Response code
Comments:	This function is identical to *TransformObject* except the object's fill is recalculated after the transform operation.
Also see:	*GetKFactor, GetJobStatus, TakeHostControl, GetObjectType, GetObjectRect, GetObjectCenter, ResetObjectTransform, SetObjString, TransformObject,TransformObjectByName*

TransformObjectByName	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Applies rotation, scaling, offset and (optionally) a new string value to the specified named string based object and concatenates the transform values to the current *UserTransform*.
Implementation:	"**113**,objectname,newstring,rotation,rotationcenterx,rotationcentery,xscale,yscale,xoffset,yoffset"
Parameters	objectname: The name of the object. Valid number of characters: [1 to 255]
	newstring: The new string value. There cannot be embedded commas in the string. To pass strings with embedded commas, see *SetObjectString* or *TransformObjectByNameEx*. Valid size: [1 to 2999 characters] Special conditions: *DataMatrix*: To imbed control characters in the string, use the tilde (~) character before the control code. To imbed an actual ~ character, use two tilde characters in a row (~~). Exception: To imbed an ASCII 0 character (NUL) , use ~@ instead of the ASCII 0.
	rotation: The relative amount to rotate the object, in degrees. A positive rotation value results in a clockwise rotation. Valid range: [-360.0 to 360.0]
	rotationcenterx: The coordinate position representing the center of rotation in the x-axis. This value can be a coordinate position that is outside the normal marking field. Valid range: [-2,147,483,648 to 2,147,483,647] Units: Firmware 2.x [bits] Firmware 6.x, 7.x [µm]
	rotationcentery: The coordinate position representing the center of rotation in the y-axis. This value can be a coordinate position that is outside the normal marking field. Valid range: [-2,147,483,648 to 2,147,483,647] Units: Firmware 2.x [bits] Firmware 6.x, 7.x [µm]
	xscale: Amount to scale object in the x-axis. Valid range: [> 0]
	yscale: Amount to scale object in the y-axis. Valid range: [> 0]
	xoffset: The amount to move the object in the x-axis. Valid range: [-2,147,483,648 to 2,147,483,647] Units: Firmware 2.x [bits] Firmware 6.x, 7.x [µm]
	yoffset: The amount to move the object in the y-axis. Valid range: [-2,147,483,648 to 2,147,483,647] Units: Firmware 2.x [bits] Firmware 6.x, 7.x [µm]
Returns:	An API Response code
Comments:	Client must call *TakeHostControl* before making this call. The Object at *objectindex* must be a valid marking object type. The transformations are applied in the following order: 1. String is changed 2. Rotation 3. Scaling 4. Offset

TransformObjectByName (cont.)	
Comments (cont.):	Both the Object *outline* and the Object *fill* are transformed with this call. When calling *SetObjectString*, or if the object will change its string value at run-time because of serialization, AutoDate, etc. the Object is processed in the following order: 1. New Outline vectors are generated 2. The JobTransform is applied (from the Job) 3. The Object is Justified 4. New Fill vectors are generated 5. The *UserTransform* is applied Subsequent calls to *TransformObject* are cumulative, as each transform is concatenated to the current *UserTransform*. To clear all transforms (setting the *UserTransform* to the Identity matrix), call *ResetUserTransform*. In order to maximize performance, the Remote API does not check whether an objects vector list is within the legal marking field. Therefore, **it is the responsibility of the programmer** to insure that after an object has been transformed, it is within the legal marking field boundaries. Undefined results can be expected if an attempt is made to execute an object with a vector list outside the legal marking field. The bounds of an object can be discovered by calling *GetObjectRect*. The *marking field* is described using a Cartesian coordinate system, with: Center: [(0,0)] Bottom Left: Firmware 2.x [(-32768, -32768)] bits Firmware 6.x, 7.x [(-fieldsize / 2, -fieldsize / 2)] µm Top Right: Firmware 2.x [(32768, 32768)] bits Firmware 6.x, 7.x [(fieldsize / 2, fieldsize / 2)] µm
Also see:	*GetKFactor, GetJobStatus, TakeHostControl, GetObjectType, GetObjectRect, GetObjectCenter, ResetObjectTransform, SetObjString, TransformObject, TransformObjectByNameEx*

TransformObjectByNameNewFill	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Applies rotation, scaling, offset to the object outline, generates new fill, (optionally) applies a new string value to the specified named string based object, then concatenates the transform values to the current *UserTransform*.
Implementation:	"**128**,objectname,rotation,rotationcenterx,rotationcentery,xscale,yscale,xoffset,yoffset,newstring"
Parameters	See *TransformObjectByNameEx* for parameter details.
Returns:	An API Response code
Comments:	This function is identical to *TransformObjectByNameEx* except the object's fill is recalculated after the transform operation.
Also see:	*GetKFactor, GetJobStatus, TakeHostControl, GetObjectType, GetObjectRect, GetObjectCenter, ResetObjectTransform, SetObjString, TransformObject, TransformObjectByName*

TransformObjectByNameEx	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Applies rotation, scaling, offset and (optionally) a new string value to the specified named string based object and concatenates the transform values to the current *UserTransform*.
Implementation:	"**114**,objectname,rotation,rotationcenterx,rotationcentery,xscale,yscale,xoffset,yoffset,newstring"
Parameters	See *TransformObjectByName* for parameter details.
Returns:	An API Response code
Comments:	This function is identical to *TransformObjectByName* except the *newstring* parameter is placed last in the command string. This allows embedded commas in *newstring*.
Also see:	*GetKFactor, GetJobStatus, TakeHostControl, GetObjectType, GetObjectRect, GetObjectCenter, ResetObjectTransform, SetObjString, TransformObject, TransformObjectByName*

TurnLaserOff	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Immediately switches the laser signals to the OFF state, as defined in the currently loaded laser configuration file.
Implementation:	"**58**"
Parameters	None
Returns:	An API Response code
Comments:	The ActiveJob must be *Idle* for this command to succeed. *TurnLaserOff* is a *Control Command*, and the instructions are placed in the FIFO immediately.
Also see:	*TurnLaserOn*

TurnLaserOn	**Supported Platforms: Firmware Code 2.x, 6.x and 7.x**
Purpose:	Immediately switches the laser signals to the ON state, as defined in the currently loaded laser configuration file
Implementation:	"**57**,digital,analog1, analog2,frequency,pulsewidth,lasertick,duration"
Parameters	digital: The value to set the 8-bit Laser Power Data port to. Valid range: [0 to 255]
	analog1: The value to set the 12-bit Laser Analog Power output to. The actual analog voltage output will depend on the configuration of the output range. Valid range: [0 to 4095]
	analog2: The value to set the 12-bit Laser AOM output to. Valid range: [0 to 4095]
	frequency: The frequency, in kHz, of the Laser Mod 1 and Laser Mod 2 signals. The range of valid frequencies depends on the value of *lasertick*. lasertick = 5, valid range: [0.2 to 5000.0 kHz] lasertick = 50, valid range: [0.02 to 900.0 kHz]
	pulsewidth: The pulse width, in microseconds, of the Laser Mod 1 and Laser Mod 2 signals. The range of valid pulse widths depends on the value of *lasertick*. lasertick = 5, valid range: [0.1 to 5000.0 µs] lasertick = 50, valid range: [1.0 to 5000.0 µs]
	lasertick: Timing value that controls the available range of frequencies and pulse widths. See *frequency* and *pulsewidth* (above) Valid values: [5 or 50]

TurnLaserOn (cont.)	
Parameters (cont.)	duration: The time, in micro-seconds, to fire the laser. A value of zero will turn on the laser indefinitely, or until a *TurnLaserOff* command is received. Valid range: [0 to 2147483648 µs)
Returns:	An API Response code
Comments:	The ActiveJob must be *Idle* for this command to succeed. *TurnLaserOn* is a *Control Command*, and the instructions are placed in the FIFO immediately.
Also see:	*TurnLaserOff*

API Response Codes

The following are possible responses returned by the API. In certain cases, the response message may be an error message rather than the expected *Success* or return variable(s).

API Response codes

Value/Constant	Description
0 = Success	The operation completed successfully
1 = Idle	The job engine is idle
2 = Busy	The job engine is currently executing a job
3 = NoJob	The specified job was not found
4 = InControl	The requesting client has exclusive control of the Host
5 = NotInControl	The requesting client does not have exclusive control of the host
6 = LicenseUnavailable	A valid license was not found
7 = LicenseAccessDenied	The current license does not allow the requested feature
8 = BadCommand	The API command was not recognized
9 = BadArg	A specified argument was invalid
10 = ArgOutOfRange	A specified argument was out of range
11 = UnknownTimeZone	The specified time zone cannot be found
12 = Reserved	Reserved
13 = BadConversion	Error while converting between multi-byte and Unicode characters
14 = RegistryError	A Windows CE Registry read or write operation failed
15 = TimeZoneFileError	A Time Zone File operation failed
16 = ResetInterlock	An interlock was signaled and must be reset by calling Abort
17 = ListNotOpen	An operation was attempted on a list that has not been opened
18 = ListAlreadyOpen	The list is currently open
19 = BadData	The data in the specified file was not in the correct format
20 = APIException	The Remote API caused an unexpected exception
21 = JobAborting	The job is currently aborting from a previous abort command
22 = FPGALoadFile	An attempt to load the FPGA with instructions failed
23 = JobManagerInitFail	The Job Manager failed to initialize properly
24 = LaserLoadFail	The specified laser configuration failed to load properly
25 = LensLoadFail	The specified lens configuration failed to load properly
26 = PMLoadFail	The specified Performance Matrix configuration failed to load properly
27 = MotionLoadFail	The specified motion configuration failed to load properly
28 = HostManagerInitFail	The Host Manager failed to initialize properly
29 = InvalidIPAddress	The specified IP address is not a valid IPv4 IP address
30 = DataUnknown	The format of the data is not recognized

API Response codes (continued)

Value/Constant	Description
31 = BadChecksum	The data failed a checksum test
32 = NetworkShareNotConnected	There was an attempt to use a network resource, but no connection exists
33 = NetworkConnectFail	An attempt to connect to a network share failed
34 = UnknownNetworkError	An unspecified network error has occurred
35 = APICommandTimeout	A command that was sent to the Remote API timed out
36 = ExternalProcessFail	Internal use
37 = DLLLoadFail	Internal use
38 = NoAdapter	No Network adapter was found
39 = AddIPAddressFailure	An attempt to add a temporary IP address failed
40 = BadAPIResponse	The Remote API returned an unexpected response
41 = CannotCreateSocket	Internal use
42 = CannotConnectSocket	Internal use
43 = CannotGetFPGABufInfo	Internal use
44 = CannotGetFPGABuf	Internal use
45 = CannotWriteFPGABuf	Internal use
46 = FPGAException	Internal use
47 = FTPConnectionError	An attempt to connect to an FTP resource failed
48 = FileAlreadyExists	The specified file already exists in the specified location.
49 = UnknownOS	Internal use
50 = SocketException	Internal use
51 = ProcessTimeout	The process returned a time out error
52 = DeviceNotFound	The specified device cannot be found
53 = LoginInProcess	An attempt to connect to a device is currently executing
54 = APIClientInControl	There is an active connection to a Remote API client
55 = StreamClientInControl	There is an active connection to a WinLase client
56 = CannotConnectToAPI	An attempt to connect to a device using the Remote API port failed
57 = ReadFail	Internal use
58 = StreamBufferFull	Internal use
59 = NoConfigRecord	Cannot find the specified configuration record
60 = OperationCanceled	Operation was canceled by the user
61 = NoData	Internal use
62 = InitializationError	Internal use
63 = FailToCreateServiceThread	Internal use
64 = CannotOpenDevice	Internal use
65 = SegmentFull	Internal use
66 = MarkerLibraryNotInitialized	An operation was attempted before the Marker Library was initialized
67 = RingBufferNotInitialized	An operation was attempted before the Ring Buffer was initialized

API Response codes (continued)

Value/Constant	Description
68 = AccessDenied	Access to a resource was denied
69 = RequiresUACElevation	The attempted operation can only be performed by elevation of the UAC
70 = NotAllowed	The requested operation is not allowed
71 = NoLaserConfig	The laser config file was not found
72 = NoLensConfig	The lens config file was not found
73 = OutOfMemory	There is not enough memory to complete the task
74 = LensTableNotFound	The specified lens correction table cannot be found
75 = HostControlIntError	No Host Controllers loaded during device boot time
76 = NoBytesRead	A read operation failed with no bytes read
77 = WritePending	Data was added to the Pending queue and will execute when possible.
100 = NoFilesFound	No files were fund at the specified path
101 = NoDrive	No drive was found
102 = JobOutOfMemory	Out of memory exception
103 = TooManyObjects	Internal error, consult factory
104 = NoObject	The specified object does not exist
105 = JobException	An internal job exception
106 = NotInHostControl	Operation cannot be performed if the client is not in control
107 = WrongHostType	Operation cannot be performed with this host type
108 = ErrorJobBusy	Operation cannot be performed while a job is executing
109 = NoActiveJob	There is no Active Job
110 = ErrorSoftware	Internal error, consult factory
111 = LoadFail	A job load failed
112 = NoObjects	Job file version not compatible with current firmware
113 = WriteFail	Internal error writing job file
114 = JobFileFormat	Job file format error
115 = FileException	Internal error while processing file
116 = UnknownObject	Unknown object type
117 = UnknownType	Unknown type
118 = NotSupported	Operation not supported
119 = NotAvailable	Resource not available
120 = FPGADataFail	Internal FPGA data format failure
121 = FileNotFound	The file specified was not found
122 = FileCreationError	Error while attempting to create a file
123 = WriteFileFail	Not all data was written to the file
124 = PathNotFound	The specified path was not found
125 = NotInCacheMode	The command requires that the job was started in Cache mode
126 = NotWaitingForStartMark	The FIFO is not currently waiting for a Start Mark signal

API Response codes (continued)

Value/Constant	Description
127 = MotionNotHomed	The command is not allowed if the device is not homed
128 = No3DModel	The operation cannot complete because there is no 3D model loaded
129 = ProjectionError	An error was encountered while projecting onto the 3D model
200 = NoProperties	Object does not contain any properties
201 = ObjectException	Internal object exception
202 = Abort	Operation was aborted
203 = NoFontResource	The font specified in the object was not found
204 = NoOverride	Internal object error
205 = ExternalEnableDenied	Operation denied by External control
206 = CannotCreatePort	A port setting (baud rate, stop bits, etc.) is invalid
207 = CannotOpenPort	Error while attempting to open COM port
208 = PortNotOpen	Port must be open to execute command
209 = PortTimeout	A port operation timed out
210 = WrongPortNumber	Invalid port number
211 = WrongObjectType	Operation is not supported by this object type
212 = AxisNotConfigured	A motion control axis referenced in an object has not been configured
213 = TextBufferOverrun	Too many characters in buffer (MAX 3000 including command opcode)
214 = InvBarcodeStringValue	The barcode string to encode contains invalid characters
215 = InvBarcodeStringLength	The length of the barcode string to encode is either too short or too long
216 = InvBarcodeNarrowWidth	The Narrow to Wide ratio is invalid
217 = InvBarcodeWidthReduce	The Width Reduction value is invalid
218 = InvBarcodeECC	The ECC value is invalid, or cannot be used to encode the string
219 = BarcodeOutOfMemory	There was insufficient memory to complete an internal barcode operation
220 = BarcodeUnknownError	Undocumented error. Please notify the factory
221 = BarcodeException	Incorrect data, or an internal operation resulted in an unexpected result
222 = NoVectors	An object was saved to the job with both Mark Outline and Mark Fill disabled
223 = BadMotionResponse	The motion controller responded with an unexpected value
224 = MotionDriverNotFound	An object is referencing a motion driver that does not exist
225 = AxisNotFound	An operation was attempted on an axis index that was not found
226 = EncoderNotFound	An operation was attempted that depends on an encoder, and no encoder was found
227 = InvStringValue	The string to process is not a valid string
228 = MotionControllerNotFound	The motion system cannot detect a valid controller device
229 = MotorNotProvisioned	The motor has not been provisioned for use with the LEC controller
230 = RuntimeMotionError	An error was generated during a motor move operation
231 = ObjectOutOfBounds	The object is outside the legal marking area
232 = InvVersion	Invalid version
233 = NoOutline	Not valid to have MarkOutline enabled with no outline to mark

System Error Codes

The following are System Error Codes that may be returned by the Remote API command *GetLastError*.

Value/Constant	Description
8001	QueueFull
9001	ProcessAbort
9002	FIFOEmptyTimeout
9003	EventTimeout
9004	BadOpcode
9005	FirmwareBug
9006	WriteDigitalBad
9007	SetLaserPowerBad
9008	SetCorrectionTableBad
9009	SetLaserPulseBad
9010	WaitForIOBad
9011	WaitForIOTimeout
9012	SetLaserStandbyBad
9013	CPLDTimeout
9014	LaserActiveTimeout
9015	SetMotfOrientationBad
9016	EnableMotfBad
9020	ServoFault
9021	InterlockAssert
9022	InterlockDeassert

Example Program

An example program is provided to illustrate how to initiate a session with the LEC, load a job stored locally on the controller, run the job once and then close the session.

C# Example

This is an example written for the LAN (Ethernet) interface. Error checking of the value returned from Socket.ReadLine() has been omitted for clarity.

```
//Connect to the LEC (with specific platform function call)
Socket.Connect();
string result = Socket.ReadLine(); //Welcome banner sent by LEC

//Take exclusive control of the LEC
Socket.WriteLine("2");
result = Socket.ReadLine();

//Load a job
Socket.WriteLine("205,testjob.dat");
result = Socket.ReadLine();

//Make the job active
Socket.WriteLine("201,testjob.dat");
result = Socket.ReadLine();

//Run the job once
Socket.WriteLine("207");
result = Socket.ReadLine();

//Wait while job is running
do
{
        //Check job status
        Socket.WriteLine("209");
        result = Socket.ReadLine();
        Sleep(100);
}
while (result != "Idle");

//Close session
Socket.WriteLine("3");
result = Socket.ReadLine();

//Disconnect (with specific platform function call)
Socket.Close();
```

Section II. RS-232 and TCP/IP Commands to WinLase Software (Not Remote API)

The interface provided for RS-232 and TCP/IP is textual; commands are sent over either port as ASCII text strings. The commands listed below are in support of the standard RS-232 communications.

RS-232 and TCP/IP Interface

Command Syntax

For the sake of clarity, the responses listed below each command have been listed with their descriptive error codes. In practice, the responses are returned with numerical error codes. For example; when the Host is in control of WinLase, the Interface Request Status for **STATUS,IN_HOST_MODE** will be **STATUS,512**.

Command Set

The following list describes all of the Remote Interface commands and their intended use, and is presented in alphabetical order by command name. A description of the command parameters follows each command.

HOME		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	Commands all of the motor control axes to return to their Home position. This command is valid only if WinLase has been configured with a compatible motor controller.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,NO_MOTOR_CONTROLLER	Motor controller board not found
	ERROR,MOTOR_HOME	There was an error during the homing process

MODIFY,buffer,##,******		(**Note**: WinLase does not need to be under **Host** control)
Purpose:	To store the string ****** in the internal string buffer at index ##. ## must be between 1 and 10. Text objects within the job must have their "Source" set to *Get String from Memory Buffer* to use the buffer contents. Calling this will clear the previous value stored in the buffer.	
Responses:	ACK	Acknowledged
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,NO_SUCH_BUFFER	Buffer ## out of range
	ERROR,INVALID_TEXT	The length of the ****** string is zero, or does not contain markable characters

MODIFY,field,##,******		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To modify a field of text or barcode where ## is the number of the field (object) to be modified, and "******" is the new text string. The field ## corresponds to the position the object has in the Object List within the job (i.e. the first object in the Object List would have an index value of 1). If the index values are not known at run time, use **MODIFY**, buffer instead. The marker must be OFFLINE. If the field does not exist, an error is returned.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.
	ERROR,NO_SUCH_FIELD	The ## field index is larger than the total number of objects loaded.
	ERROR,UNKNOWN_QUALIFIER	The length of the ****** string is zero, or does not contain markable characters.

MODIFY,position,##,xoffset,yoffset		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To change the position of a marking object, where ## is the number of the object to be modified, *xoffset* is the amount to move the object in bits along the x-axis, and *yoffset* is the amount to move the object in bits along the y-axis. The field ## corresponds to the position the object has in the Object List within the job (i.e. the first object in the Object List would have an index value of 1). The marker must be OFFLINE. If the field does not exist, an error is returned.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.
	ERROR,NO_SUCH_FIELD	The ## field index is larger than the total number of objects loaded.

MODIFY,rotation,##,angle		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To rotate a marking object about its center, where ## is the number of the field (object) to be modified, and *angle* is the amount to rotate the object, in units of 0.010 degrees. A positive *angle* value rotates the object clockwise. The object ## corresponds to the position the object has in the Object List within the job (i.e. the first object in the Object List would have an index value of 1). The marker must be OFFLINE. If the field does not exist, an error is returned.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.
	ERROR,NO_SUCH_FIELD	The ## field index is larger than the total number of objects loaded.

MODIFY,rotationex,##,angle,xcenter,ycenter		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To rotate a marking object about an arbitrary center of rotation, where ## is the number of the object to be modified, *angle* is the amount to rotate the object, in units of 0.010 degrees. *xcenter* is the x-axis center of rotation, in bits and *ycenter* is the y-axis center of rotation in bits. A positive *angle* value rotates the object clockwise. The object ## corresponds to the position the object has in the Object List within the job (i.e. the first object in the Object List would have an index value of 1). The marker must be OFFLINE. If the field does not exist, an error is returned		
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.
	ERROR,NO_SUCH_FIELD	The ## field index is larger than the total number of objects loaded.

MODIFY,scale,##,xscale,yscale		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To scale a marking object from its center, where ## is the number of the object to be modified, *x-scale* is the amount to scale the object in the x-axis, in percent, and *yscale* is the amount to scale the object in the y-axis, in percent. For example, to decrease the size of an object to half its current size, use the value 50.00 (%) for both the *xscale* and *yscale* values. The object ## corresponds to the position the object has in the Object List within the job (i.e. the first object in the Object List would have an index value of 1). The marker must be OFFLINE. If the field does not exist, an error is returned.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.
	ERROR,NO_SUCH_FIELD	The ## field index is larger than the total number of objects loaded.

MODIFY,zoffset,value		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	This function is intended to support AF (adjustable) focus shifter applications. The focus offset, in µm, from the nominal focus position of zero. The offset can be positive or negative, but must be within the range of the z focus optics hardware. *value* is the amount (in µm) to move the focus in the Z-direction. A negative value moves the focus towards the Z Minimum value (farthest from the lens). A positive value moves the focus towards the Z Maximum value (closest to the lens).	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.

OFFLINE		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	Commands the laser to immediately stop marking, and returns the laser to the MARKER_OFFLINE state.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,ALREADY_OFFLINE	Marker is already offline

ONLINE		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	Commands the marker to start the marking process. System will immediately start polling external start port, and enter MARKER_ONLINE state. The call automatically sets the external start flag to true, and sets the repeat mode to repeat indefinitely.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,ALREADY_ONLINE	Marker is marking or waiting for external start signal.
	ERROR,NO_JOB_LOADED	No job loaded.
	ERROR,INTERLOCKS_OPEN	An interlock port on the interlock I/O card is open.
	ERROR,NO_SCANCARD	There is no scan head card installed in machine
	ERROR,NO_HARDLOCK	No Hardlock detected
	ERROR,NO_IOCARD	No I/O card installed in the computer
	ERROR,STEP_REPEAT_INVALID	The values saved in the job for step and repeat will result in an invalid object position.
	ERROR,TEXT_SOURCE_INVALID	A text object was saved with a Source value incompatible with the host interface.
	ERROR,TEXTMERGE_INVALID	There was an error while processing a TextMerge file.
	ERROR,OBJECT_OUT_OF_BOUNDS	There is an object in the job that is outside the legal marking field.

OPEN,file,######		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To open a file where "######" is a text string describing the file to be opened, and must be a fully qualified UNF file path. If the file cannot be found, or is corrupt, an error code is returned.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,FILE_NOT_FOUND	The file was not found at the indicated path location, or there was an error while opening the file.
	ERROR,UNKNOWN_VERB	First word in command line not recognized.
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized.
	ERROR,UNKNOWN_QUALIFIER	The file path was less than 3 characters in length.
	ERROR,NO_DEFAULT_NODE	When using the LEC Controller, a default node must be assigned.
	ERROR,DEFAULT_NODE_NOT_FOUND	The default node could not be found.

REQUEST,data,bits_per_mm,c#,h#		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the current value for the scan field bits/mm, where *c#* is the zero based card index and *h#* is the zero based scan head index. An example string would be: "REQUEST,data,bits_per_mm,0,0". The bits/mm value is the ratio of a point coordinate in bits and the actual position of the point in millimeters.	
Responses:	DATA,*x*	*x* is the bits/mm value.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,cyclecount		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the current cycle count. The cycle count indicates the number of full cycles, including step and repeat.	
Responses:	DATA,*x*	*x* is the current cycle count.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,partcount		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the current part count. The part count is the individual marks within a cycle.	
Responses:	DATA,*x*	*x* is the current part count.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,cycletime		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the current cycle time. The cycle time is defined as the elapsed time to do all marks within a single cycle.	
Responses:	DATA,*x*	*x* is the current cycle time.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,parttime		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the current part mark time. The part time is defined as the elapsed time to do a single mark within an overall cycle.	
Responses:	DATA,*x*	*x* is the current part time.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,jobname		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the name of the currently loaded job.	
Responses:	DATA,*filepath*	*filepath* is the fully qualified path to the currently loaded job file.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,version		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return its Version number.	
Responses:	DATA,*x*	*x* is the Version number.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,data,user		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Request WinLase to return the User currently logged on to the current Windows NT/2000 session.	
Responses:	DATA,*username*	*username* is the currently logged on user.
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized

REQUEST,field,##		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	To request data from a field of text, barcode or graphic where ## is the number of the field to be queried. The marker must be OFFLINE. If the field does not exist, an error is returned.	
Responses:	DATA,*field#,objecttype,data*	*field#* is the field# of the object.
		objecttype is the object type. Refer to API command *GetObjectType* in *Section I* for details on valid object types.
		data is the string value for text and barcodes and the graphic file path for a graphic object.
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,UNKNOWN_VERB	First word in command line not recognized
	ERROR,UNKNOWN_NOUN	Second word in command line not recognized
	ERROR,UNKNOWN_QUALIFIER	The ## field was not an integer value.
	ERROR,NO_SUCH_FIELD	The ## field index is larger than the total number of objects loaded.

REQUEST,status,interface		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Returns the current status of the Host interface.	
Responses:	STATUS,IN_HOST_MODE	Host in control of WinLase.
	STATUS,HOST_NOT_READY	Not available for host command.
	STATUS,HOST_READY	Available for host command.

REQUEST,status,lecexecution		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Returns the current status of the job execution status of the LEC controller.	
Responses:	STATUS,HOST_NOT_READY	Cannot get status of marker because host is not available for host command.
	STATUS,INTERLOCKS_OPEN	An interlock port on the interlock I/O card is open.
	STATUS,LEC_IDLE	LEC Controller is idle
	STATUS,LEC_BUSY	LEC is currently executing a job.
	STATUS,ERROR_PROCESS	There was an error while in the ONLINE mode. This error will be cleared after it is read once, and if all OK, the next response will be STATUS,MARKER_OFFLINE.

REQUEST,status,marker		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Returns the current status of the laser marker and WinLase software.	
Responses:	STATUS,HOST_NOT_READY	Cannot get status of marker because host is not available for host command.
	STATUS,INTERLOCKS_OPEN	An interlock port on the interlock I/O card is open.
	STATUS,MARKER_ONLINE	Marker is marking or waiting for external start signal.
	STATUS,MARKER_OFFLINE	Job is loaded and marker is ready to accept ONLINE command or MODIFY command.
	STATUS,NO_JOB_LOADED	No job loaded.
	STATUS,ERROR_PROCESS	There was an error while in the ONLINE mode. This error will be cleared after it is read once, and if all OK, the next response will be STATUS,MARKER_OFFLINE.

RUN		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	Commands the marker to start the marking process. System will immediately execute the currently loaded job and enter the MARKER_ONLINE state. This call ***does not*** automatically set the external start flag to true, and ***does not*** set the repeat mode to repeat indefinitely. The current job settings will be used for these two parameters.	
Responses:	ACK	Acknowledged
	ERROR,NOT_IN_HOST_MODE	WinLase must be in host mode
	ERROR,ALREADY_ONLINE	Marker is marking or waiting for external start signal.
	ERROR,NO_JOB_LOADED	No job loaded.
	ERROR,INTERLOCKS_OPEN	An interlock port on the interlock I/O card is open.
	ERROR,NO_SCANCARD	There is no scan head card installed in machine
	ERROR,NO_HARDLOCK	No Hardlock detected
	ERROR,NO_IOCARD	No I/O card installed in the computer
	ERROR,STEP_REPEAT_INVALID	The values saved in the job for step and repeat will result in an invalid object position.
	ERROR,TEXT_SOURCE_INVALID	A text object was saved with a Source value incompatible with the host interface.
	ERROR,TEXTMERGE_INVALID	There was an error while processing a TextMerge file.
	ERROR,OBJECT_OUT_OF_BOUNDS	There is an object in the job that is outside the legal marking field.

SET,control,host		(**Note**: WinLase does not need to be in **Host** control)
Purpose:	Puts WinLase into external control mode. All user input at the console is disabled.	
Responses:	ACK	Acknowledged
	ERROR,ALREADY_IN_HOST_MODE	Host is already in Host mode.
	ERROR,HOST_NOT_READY	Host cannot go into host mode because the *Allow Host Control* check box in WinLase is cleared.

SET,control,local		(**Note**: WinLase ***must*** be under **Host** control)
Purpose:	Releases WinLase from the external control mode. Enables user input at the console.	
Responses:	ACK	Acknowledged
	ERROR,ALREADY_IN_LOCAL_MODE	Host is already in Local mode.
	ERROR,MARKER_ONLINE	Marker is marking or waiting for external start signal.

Example Program

An example program is provided to illustrate how to initiate a session with WinLase, manipulate an object in the loaded job, run the job once and then close the session.

C# Example

The following pseudo-code uses an application defined function called **SENDToSocket()**, which represents a method of outputting text from either the RS-232 or TCP/IP ports, and receiving a response as it's return value.

```
//Acquire WinLase
SendToSocket("SET,control,host");

//Make sure we have control
If(SendToSocket("REQUEST,status,interface")!="STATUS,512")
        return ERROR;

//Load a job
SendToSocket("OPEN,file,c:\\test\\job\\test.wlj");

//Make sure job has loaded properly
If(SendToSocket("REQUEST,status,marker")!="STATUS,2300")
        return = ERROR;

//Change the text in the object at index position 2
SendToSocket("MODIFY,field,2,"Hello World");

//Put WinLase into the ONLINE mode, waiting for START PROCESS to toggle
SendToSocket("ONLINE");

//Verify we are in ONLINE MODE
If(SendToSocket("REQUEST,status,marker")!="STATUS,2301")
        return = ERROR;
//Mark some parts
.
.
.
//Stop polling the START PROCESS input
SendToSocket("OFFLINE");

//Release WinLase
SendToSocket("SET,control,local");
```

AMADA MIYACHI CO., LTD.

http://www.amy.amada.co.jp/e/

AMADA MIYACHI AMERICA, INC.
1820 South Myrtle Ave., Monrovia, CA 91016, U.S.A.
TEL. +1-626-303-5676 FAX. +1-626-358-8048
http://amadamiyachi.com

AMADA MIYACHI CO., LTD.
200, Ishida, Isehara-shi, Kanagawa 259-1196, Japan

AMADA MIYACHI KOREA CO., LTD.
28, Dongtanhana 1-gil, Hwaseong-si, Gyeonggi-do, 445320, Korea
TEL. +82-31-8015-6810 FAX. +82-31-8003-5995

AMADA MIYACHI SHANGHAI CO., LTD.
Room01,15th Floor, SML Center, No.610 Xujiahui Road, Huangpu District, Shanghai 200025, China
TEL. +86-21-6448-6000 FAX. +86-21-6448-6550

AMADA MIYACHI EUROPE GmbH
Lindberghstrasse 1, DE-82178 Puchheim, Germany
TEL. +49-89-839403-0 FAX. +49-89-839403-10

AMADA MIYACHI TAIWAN CO., LTD.
Rm.5, 2F., No.9, Dehui St., Zhongshan Dist., Taipei 10461, Taiwan (R.O.C.)
TEL. +886-2-2585-0161 FAX. +886-2-2585-0162

AMADA MIYACHI VIETNAM CO., LTD.
M floor, 400 Nguyen Thi Thap Street, Tan Quy Ward, District 7, Ho Chi Minh City, Vietnam
TEL. +84-8-3771-7972 FAX. +84-8-3771-7974

AMADA MIYACHI (THAILAND) CO., LTD.
40/14 Bangna Tower C, 17th Floor, Unit B, Moo 12, T.Bangkaew, A.Bangplee Samutprakarn 10540, Thailand
TEL. +66-2-751-9337-8 FAX. +66-2-751-9340

AMADA MIYACHI INDIA PVT. LTD.
Ground Floor, Raj Arcade, 5th "A" 1st Cross, HRBR Layout, Kalyan Nagar, Bangalore-560 043, India
TEL. +91-80-4092-1749 FAX. +91-80-4091-0592

AMADA MIYACHI DO BRASIL LTDA.
Av. Tamboré, 965/973, Salas P22 e F11, 06460-000, Barueri, SP, Brasil
TEL. +55-11-4193-1187

AMADA MIYACHI AMERICA, INC.
1820 South Myrtle Ave., Monrovia, CA 91016, U.S.A.
TEL. +1-626-303-5676 FAX. +1-626-358-8048