EC1000™ Controller

Advanced Laser Positioning & Control
for Laser Steering Systems

OEM Integrators Manual

Revision 1.5.1

Cambridge Technology

25 Hartwell Avenue, Lexington, MA 02421
Tel. 781-541-1600   Fax. 781-541-1601
www.cambridgetechnology.com

CAMBRIDGE TECHNOLOGY, INC.
25 Hartwell Avenue
Lexington, MA 02421-3102
U.S.A.
TEL.781-541-1600
FAX.781-541-1601

Lit. No. P0900-0120   Rev. 1.5.1   8 Feb 2008

Table of Contents

Table of Contents

Table of Contents

1 Introduction

1.1 General Notes

Cambridge Technologies reserves the right to make changes to the product covered in this manual to improve performance, reliability or manufacturability.

Although every effort has been made to ensure accuracy of the information contained in this manual, Cambridge Technology assumes no responsibility for inadvertent errors. Contents of the manual are subject to change without notice.

1.2 Using this manual

1.2.1 Purpose

This manual provides the following information for the EC1000 system:

- product description, installation, operation & troubleshooting

1.2.2 Scope

This manual covers the EC1000 Control Board and the EC1000 I/O Board only.

1.2.3 Revision History

REVISION	DATE	Changes from previous revision
0.1	19 December 2005	Second review draft
0.2	16 February 2006	Third review draft
0.3	06 March 2006	Major update in intro sections; software API update to include MOTF functions, etc.
0.4	15 March 2006	Major update with revamped laser timing and reordered API presentation
0.5	25 April 2006	Corrected coordinate diagram; refined XML for laser config and controller config; reorganized sections; updated table of contents; updated all laser timing diagrams; fixed various connector diagrams and tables
1.0	27 April 2006	Initial Release
1.1	29 June 2006	Added sections on API installation and usage; corrected some inconsistancies in fixed data XML tables; added XML examples for configuring the various laser modes; added timing diagram for fiber lasers; fixed EC1000-IO module connectorization diagram errors; added StartJob and EndJob job tags.
1.1a	5 July 2006	Fixed connector ID for EC1000 power connector on page 86: was J6, is now J20
1.2	28 November 2006	Changed API function names to use Microsoft interface naming conventions Added argument to Broadcast API to permit specifying a particular network adaptor Added interface definition for OnConnect events Changed syntax of PriorityMessage XML Updated AdminConfig file to include Broadcast IP information and Head SN information Fixed error in Figure 19 where pin 13 of J13 was erroneously called pin 14 Added notation about Rev Cx of the I/O Module to Figure 5
1.3	21 May 2007	Updated CTI address and contact information Reorganized document sections to present installation sections first Fixed signal naming inconsistancies on several connector diagrams Fixed signal conditioning diagrams to correctly illustrate digital input handling Fixed XML syntax of MotfRestJump Fixed units of MarkDelay and PolyDelay Changed syntax of Abort priority message Hardlock licensing of the API has been removed Added section on pendant control and remote control protocol Changed syntax of certian sections of the AdminConfig file Updated API installer section

1.4	21 November 2007	SysInfoData expanded to include object library version info and USB storage info SysStatusData expanded to include the current state of the digital I/O signals AdminData is now accessible via the request/sendFixedData methods Hexidecimal data format use is more explicitly specified Added LaserConfig parameters to enable IPG Laser DLATCH and GUIDELASER support Correction table XML data can now be sent using the sendStreamData method The Draw job command was added to facilitate rapid re-drawing of preview images. The LaserOff job command was removed since it was a duplicate of the LaserSignalOff command New session methods were added to support job management Job managment methods were expanded to include the specification of storage location and name Clarified behavior of the system under exception conditions Restart was added to the Priority Data Definitions Exception reporting via message events was clarified The COMWriteLine remote command was expanded to include waiting for a device response The maximum value for various timeout values specifed in the job commands have been corrected
1.5	18 January 2008	Added support for Wobble Fixed units on some of the LaserConfigData timing parameters Clarified behavior of the system on an interlock trip Minimum stepPeriod for marks and jumps reduced to 10usec Added support for four correction tables enabling fully functional dual-headed system operation WaitForIO now supports indefinite waiting as a special case Added job name argument to the remote commands ExecuteJobOnce and ExecuteJobContinuous Clarified synchronization requirements for ExecuteJobOnce and ExecuteJobContinuous LaserPower now only affects the 8-bit digital power port. WriteAnalog is used to affect the analog power port.
1.5.1	7 February 2008	Corrected syntax of WobbleEnable command Corrected comments about the LaserPower command affecting the analog output

1.3 Obtaining Technical Assistance

If you encounter a problem:

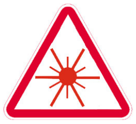
1. Review all of the information contained in this manual.
2. Consult your own internal people about the issue.
3. If you need further assistance, call Cambridge Technology, Monday through Friday, 9 A.M. until 4:30 P.M. (Eastern Standard Time) at 781-541-1600.

2 Safety

Please read all operating instructions completely before installing and using the EC1000 boards.

2.1 Safety labels and symbols

The following safety labels and symbols are used throughout the system documentation:

Label	Meaning
DANGER	Serious bodily injury or death.
⚠ WARNING	Potential for serious bodily injury.
⚠ CAUTION⚠	Potential for property damage and/or minor bodily injury.
	SHOCK HAZARD Electrical voltage present. Take appropriate measures to protect yourself from electrical shock.
	LASER HAZARD.

2.2 General safety guidelines

DANGER

Laser Radiation
Do not stare directly into a laser beam.
Follow all system laser safety requirements during installation and operation.

DANGER

Laser Radiation
Cambridge Technologies recommends the use of a shutter to prevent unwarranted emission of laser radiation, where practical.

2.3 SAFETY/CAUTIONS

⚠ WARNING

Use of controls, adjustments, or procedures other than those specified in this manual without consulting a competent safety professional may result in component damage, and/or exposure to potential hazards. Always follow established industrial safety practices when operating equipment.

⚠ CAUTION⚠

ESD HAZARD!
Use appropriate anti-static wrist straps and/or work area equipment to prevent damage to the board electronic components.

⚠ CAUTION⚠

Always check your application program BEFORE running it. Errors can cause system damage.

⚠ CAUTION⚠

Electronic boards are fragile! Handle and store with care.
Protect electronic components from dust, humidity, electromagnetic fields, static electricity, chemicals, and mechanical stress.

3 EC1000 Product Introduction

3.1 EC1000 System Description

The EC1000 is a self-contained controller that provides advanced hardware and software control technology to drive laser scanning systems. The EC1000 control board is specifically designed for remote embedding and control of a scan head or laser system, and is capable of controlling up to three motion axes with concurrent laser timing control. It also provides integrated synchronization I/O for connection to factory automation equipment.

Connection to a PC for job download and administrative control is made via Ethernet® network using industry standard TCP/IP protocols. In addition to Ethernet connectivity, the EC1000 provides external USB connections to support job file distribution via industry standard USB FLASH disks. RS232 Serial I/O is also provided for a pendant style user interface, serial laser control, and diagnostic access.

An optional I/O board provides an off-the-shelf solution for communication and power connectivity, or custom cabling configurations as desired. In a typical installation, the EC1000 is an "embedded" device, installed remotely in a laser scanning system. Positioning vectors are streamed from a networked PC to the remote EC1000 board which processes these vectors in real-time and sends them to the laser steering galvo servos as analog or digital signals. Alternatively, the vector stream can originate from a locally stored file in on-board or external USB based FLASH memory.

There is no requirement to dedicate a full-time host PC to a laser scanning system, as the EC1000 board can process vectors while the PC is used for other purposes. In fact, one PC can support multiple EC1000 based scanning systems with no loss in performance. This is due in part to the large amount of buffer memory available on the controller, the use of a separate supervisory processor on the controller to handle network communication processing, and the complete off-loading of time critical tasks to a second real-time processor on the EC1000.

3.2 EC1000 Features

3.2.1 Hardware features

- stand-alone design targeted at "embedded" installation in scanning equipment
- dual processor architecture with integrated 10/100BaseT Ethernet communication capability
- real-time processing engine for precise, synchronized scanner movement and laser control
- fully programmable laser control signals for commonly used lasers
- direct analog or digital interface to XY or XYZ scan head galvanometer servo controllers
- 16-bit galvanometer position command resolution
- integrated lens distortion correction table support
- integrated slave head control via XY2-100 standard protocol
- software selectable polarity and timing of all laser control signals
- two auxiliary analog output channels (12-bit) 0-10V for control of laser current or pulse intensity
- one 8-bit optically isolated digital output port for laser power control
- optically isolated digital inputs and outputs (four each) for external equipment synchronization
- four optically isolated interlock inputs
- two USB host ports for portable FLASH disk access and other peripheral I/O
- 16Mbytes of on-board FLASH for local job and parameter storage
- one RS232 serial pendant port
- one RS232 serial laser control port
- two RS232 diagnostic control ports

3.2.2 Software features

The EC1000 is designed to fit into a client-server architecture. The module implements all required server code functions including identification broadcast, data streaming, command and control communications, and real-time positioning operations. Host to module communications uses TCP/IP as a transport mechanism over Ethernet.

To simplify integration with third-party application software, a Windows .NET/COM Application Programming Interface (API) to the functions of the module is provided. The API takes care of all of the network connection requirements and abstracts many of the discrete functions of the module into higher level vector oriented instructions.

Key features supported by the software are:

- compatibility with Windows® 2000, XP, and ME operating systems
- administrative management of the EC1000 including scanning head configuration data
- automatic device recognition for any number of network attached controllers
- COM and .NET Assembly access to all scanning functions using XML for parameter and data passing
- support for lens correction files (65 x 65 data points) to correct for field distortions
- designed to be easily integrated with custom or standard commercial marking software applications

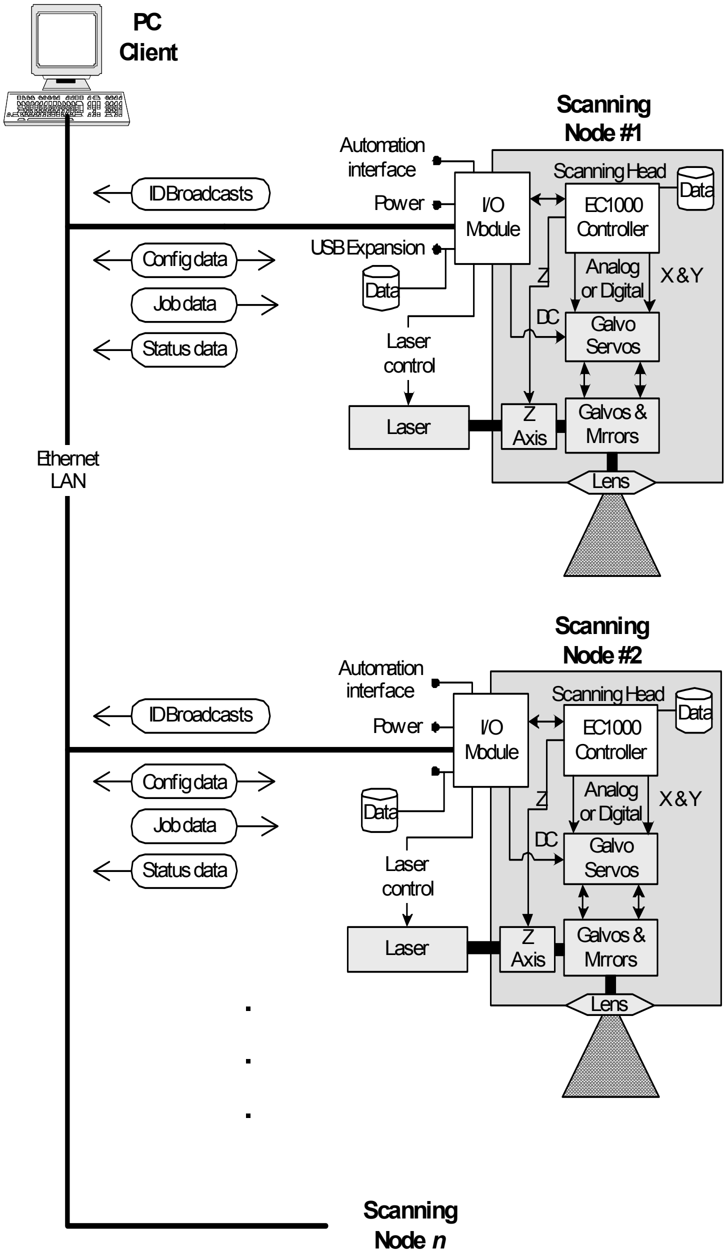


Figure 1 EC1000 Control Board typical installation

3.3 Technical Specifications

Category	Feature	Specification
Galvo control	Number of axes	3 (X, Y, Z)
	Position command resolution	16-bit (-32768 to +32767)
	Position command output	Analog • Differential output with software programmable range: +/- 2.5 volts +/- 5.0 volts +/- 10 volts • Optional jumper configuration provides additional range capability of: +/- 1.5 volts +/- 3.0 volts +/- 6.0 volts Single ended operation possible using an analog ground reference and one of the differential output signals. Signals present a 20 Ohm source impedence. Digital • Full-time SPI serial digital output for the X&Y axes, 3.3V TTL compatible. Signal values represented at the analog outputs are reflected on the SPI data channels. • Factory jumper configurable SPI serial digital output for the Z axis, 3.3V TTL compatible. Signals are presented at the Z axis connector in lieu of the analog signals. • XY2-100 compatible protocol for X, Y and Z axes including status read back. Signal values represented at the analog outputs are reflected on the XY2-100 data channels.
	Auxiliary signals	Motor enable ouputs, per axis, 5 volt TTL compatible, programmable polarity Motor status inputs, per axis, 5 volt TTL compatible, programmable polarity
Laser control	Number of signals	6, software programmable polarity and timing • LASERENABLE: asserted a programmable time prior to a sequence of mark instructions and de-asserted after a programmable period of laser inactivity • LASERON1: asserted when the laser is active • LASERPTR: laser pointer control signal • LASERFPK: programmable laser first pulse killer, or suppression signal • LASERMOD1: programmable laser modulation or Q-switch pulse stream • LASERMOD2: second programmable laser modulation or Q-switch pulse stream, 180 degrees phase shifted from LASERMOD1
	Electrical output	5 volt TTL compatible
	Time base resolution	20ns
Automation interfaces	User inputs	4, optically isolated, programmable polarity and level or edge sensitivity A marking job may contain an instruction that pauses execution until one of these signal is asserted by external equipment
	User outputs	4, optically isolated, programmable polarity A marking job instruction may specify the state of any of these signals
	System status	3 signals, optically isolated • BUSY: asserted then a BeginJob instruction is executed and de-asserted when an EndJob instruction is executed • MARKINPRG: asserted when marking is in progress • ERROR: asserted if an error is detected
	System synchronization	1 signal, optically isolated, programmable polarity and level or edge sensitivity • STARTMARK: a marking job may contain an instruction that pauses execution until this signal is asserted by external equipment
	Conveyor tracking	RS422 digital quadrature inputs (A & B phases + Index) Used for tracking objects in motion and automatically compensating for that motion while marking. Also known as marking-on-the-fly. Compensation can be software configured to be applied to either the X or Y axis
Safety	Interlock protection	4, optically isolated, programmable polarity Used to provide hardware level protection from accidental exposure to laser radiation
Communication	Ethernet	10/100 BaseT compatible
	Serial RS232	1 full modem interface capable, typically reserved for pendant style user interface access 1 three-wire (Tx + Rx + Gnd) port for diagnostic access to main processor 1 three-wire port for diagnostic access to marking engine processor 1 three-wire port for serial communication to a laser under job control
Peripherals	USB	2 USB host ports for access to external Flash memory disks or other peripherals
Electrical	DC Power	+/- 15-28 volts @ 150ma +5 volts @ 750ma
Mechanical	Length x Width	5.0” x 4.0”
	Mounting holes	0.125” diameter located 0.156” from each corner

4 Installation Requirements/Precautions

4.1 Storage and Installation Environment

⚠ **CAUTION** ⚠

ESD HAZARD!
Use appropriate anti-static wrist straps and/or work area equipment
to prevent damage to the board electronic components.

Protect the EC1000 boards from mechanical stress, humidity, dust, and thermal damage. Storage temperature is -20° C to + 60° C. Operating temperature is 15 to 40° C.

The EC1000 boards are designed for installation in or near a marking head. Remote connection/programming and download control is achieved via an Ethernet connection.

4.2 Jumper Setting for Non-standard Galvo Systems

A jumper can be installed on J30-32 to change the laser galvanometer analog command voltage from the standard (±10V, or ±5V, or ±2.5V) to ±6V, ±3V, and ±1.5V, respectively. J30 affects the X axis, J31 the Y axis, and J32 the Z axis.

Note: J30, J31, and J32 are the ONLY user jumper settings on the EC1000 board.

No Jumper = ±10V, ±5V, or ±2.5V

Jumper installed = ±6V, ±3V, ±1.5V

Figure 2 Voltage jumper setting for non-standard galvo systems.

4.3 EC1000 Board

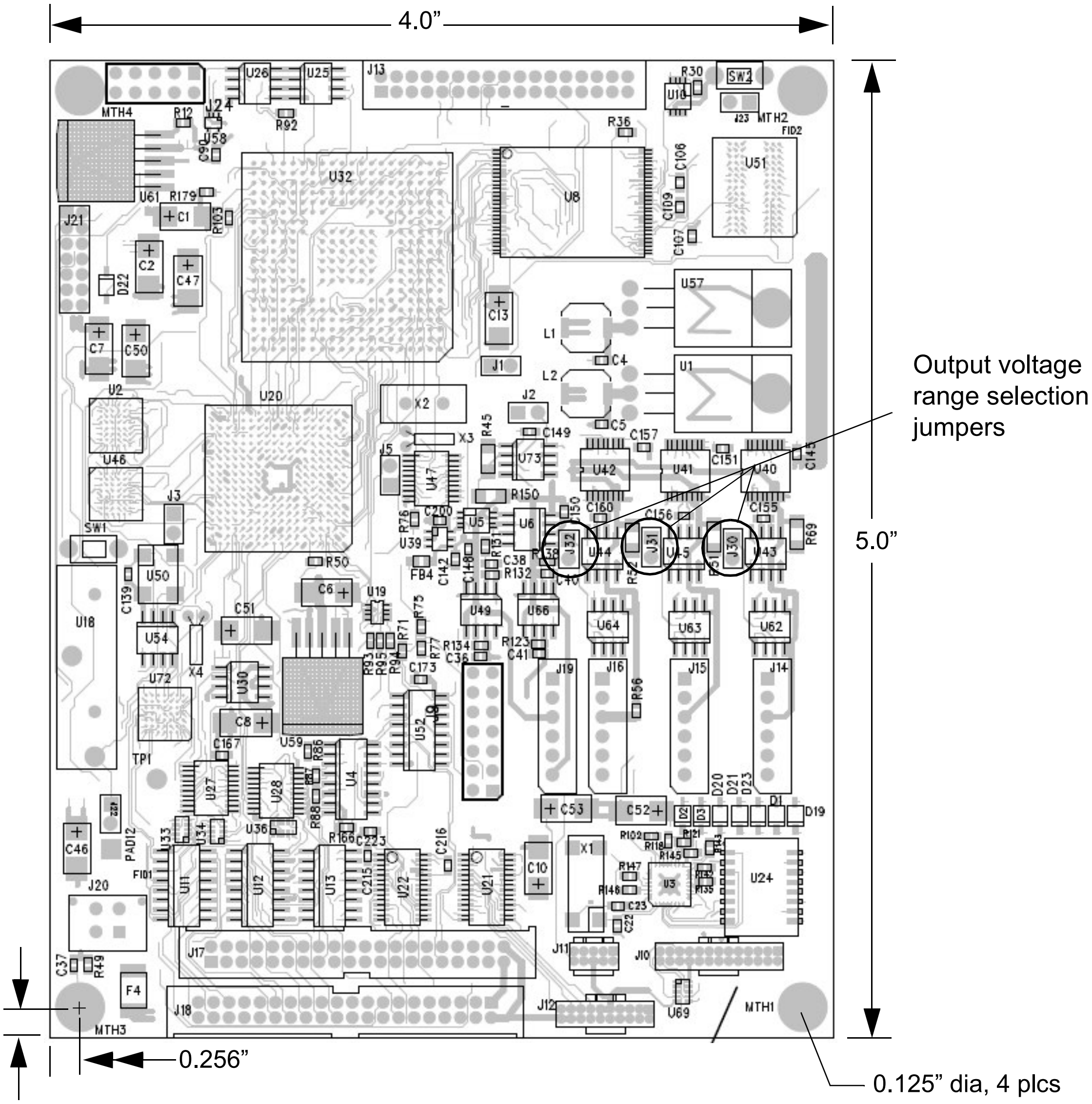


Figure 3 EC1000 Main Board Top View

4.4 EC1000 I/O Module

Because the EC1000 is a compact completely integrated controller system supporting many interface signals, it is necessary to use high-density connectors to provide access to those signals. An EC1000 I/O module provides an off-the-shelf I/O connection solution. The I/O module is designed to expose most connectors that the user will use on the top side, and all the connectors that need to connect to the main board on the back side. There are two power connectors on the back side that are intended to provide power to galvo servo controllers located inside a marker head.

4.4.1 I/O Module Block Diagram

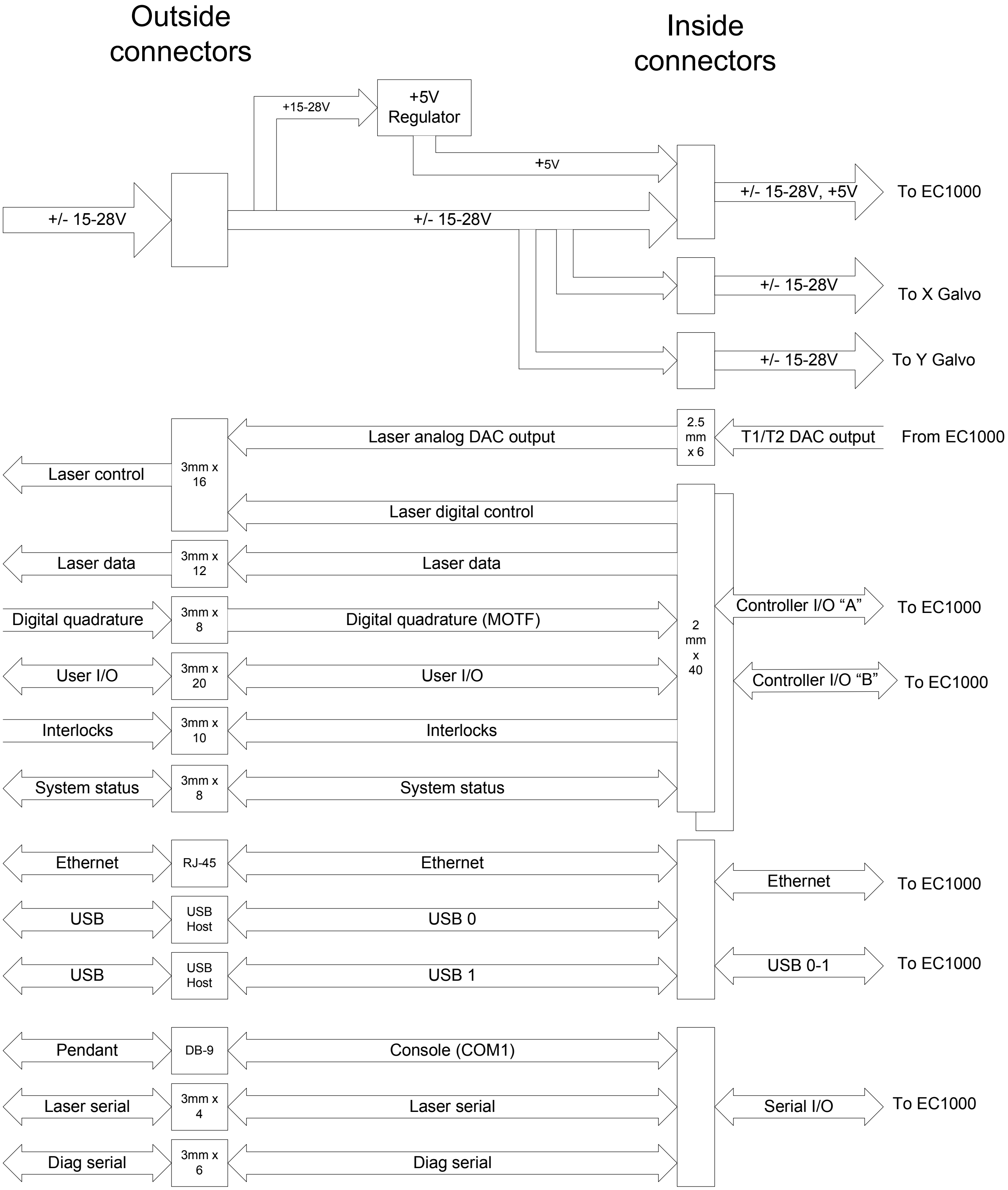


Figure 4 I/O Module Block Diagram

4.4.2 I/O Module Connectors, Rev C-E

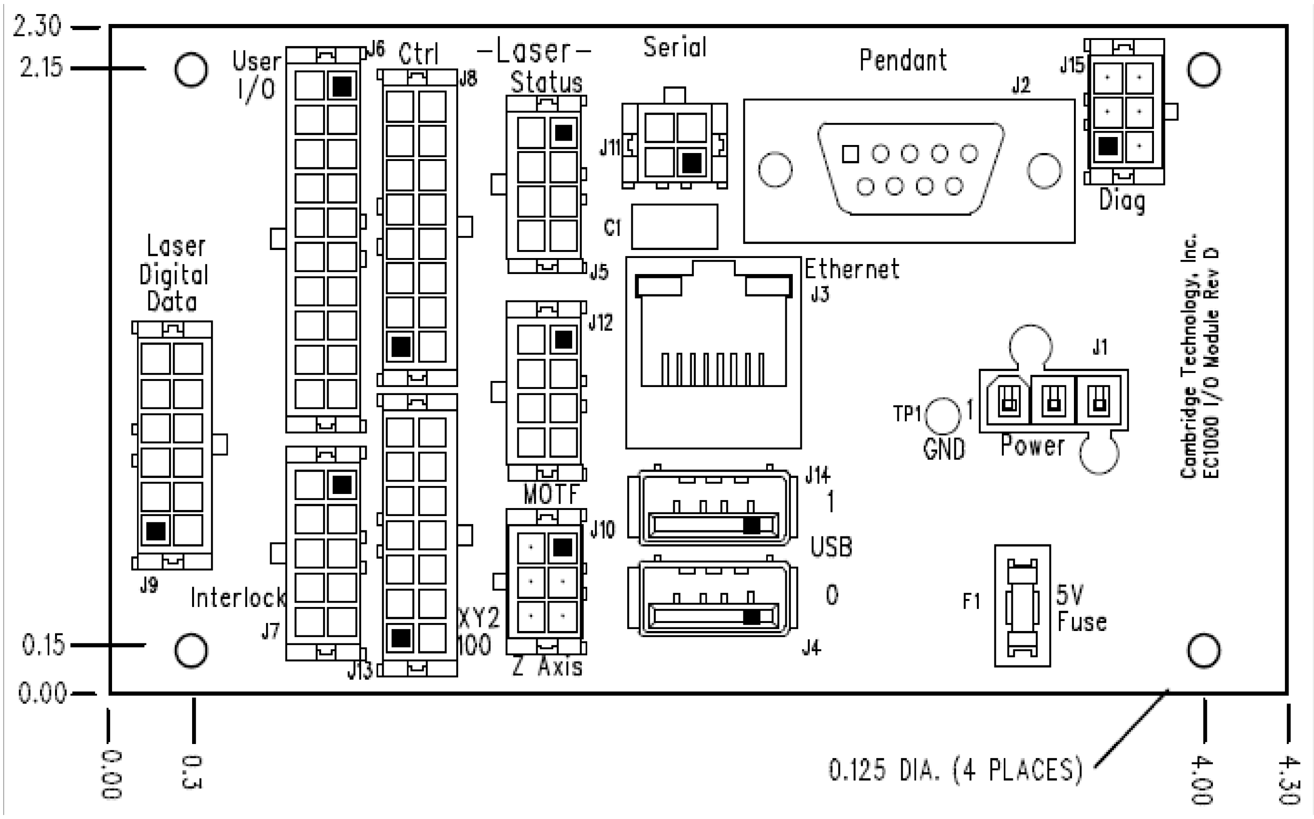


Figure 5 I/O Module Rev C-E Top View

NOTE: Rev Cx of the I/O module has connectors J11 and J15 positionally transposed from what is shown in Figure 5.

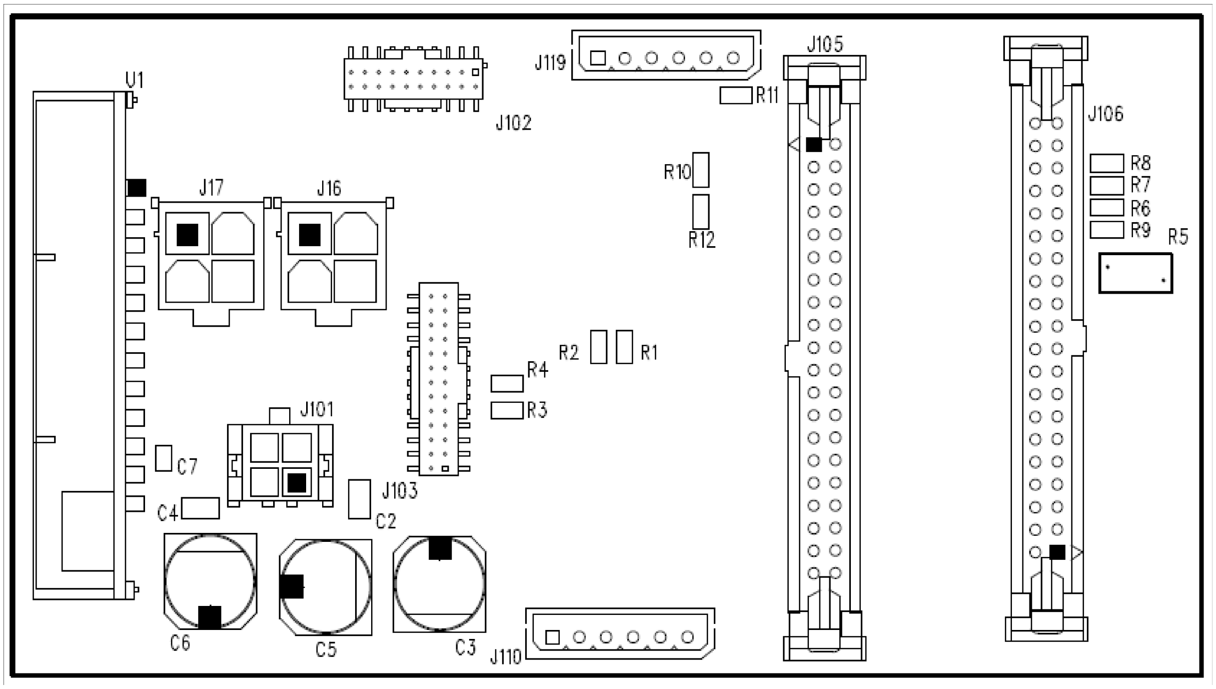


Figure 6 I/O Module Rev C-E Bottom View

4.4.3 I/O Module Connectors, Rev F

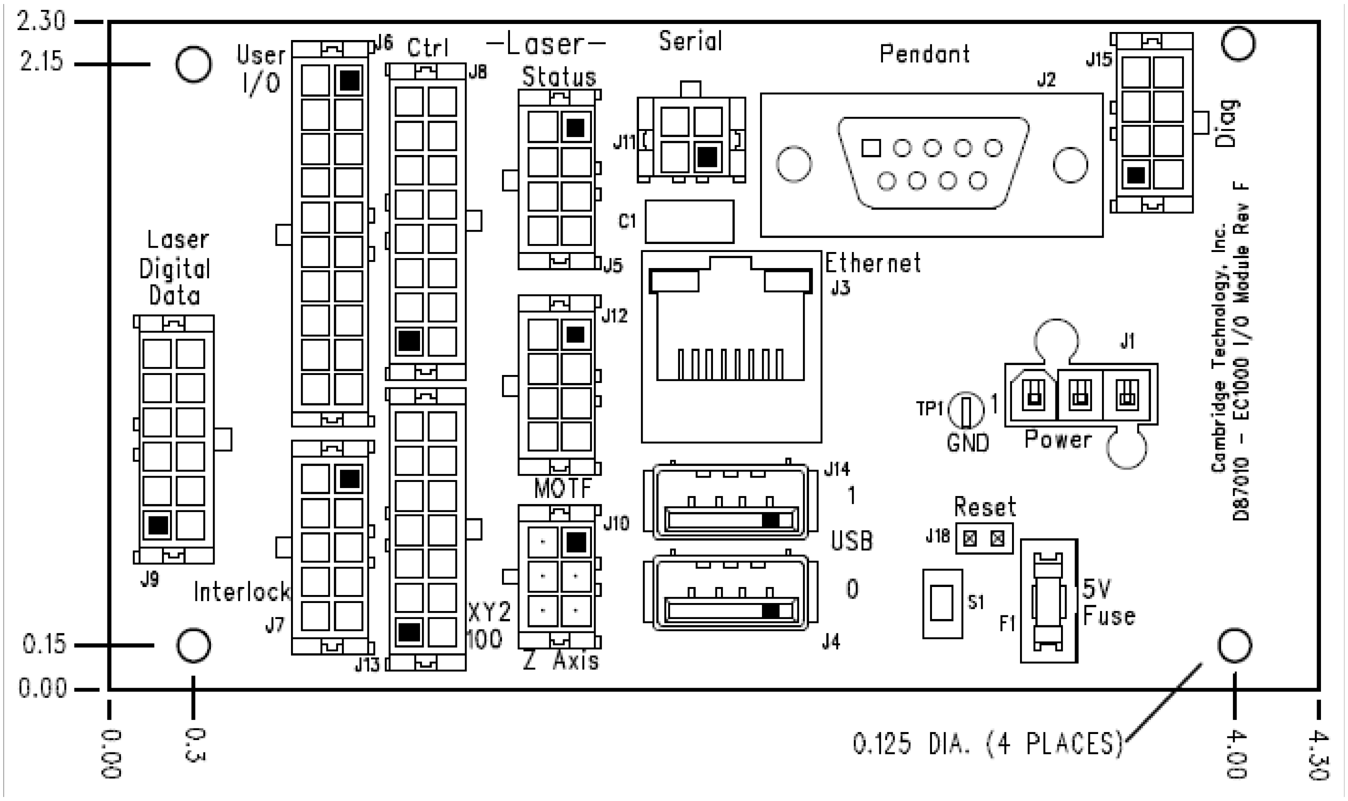


Figure 7 I/O Module Rev F Top View

.

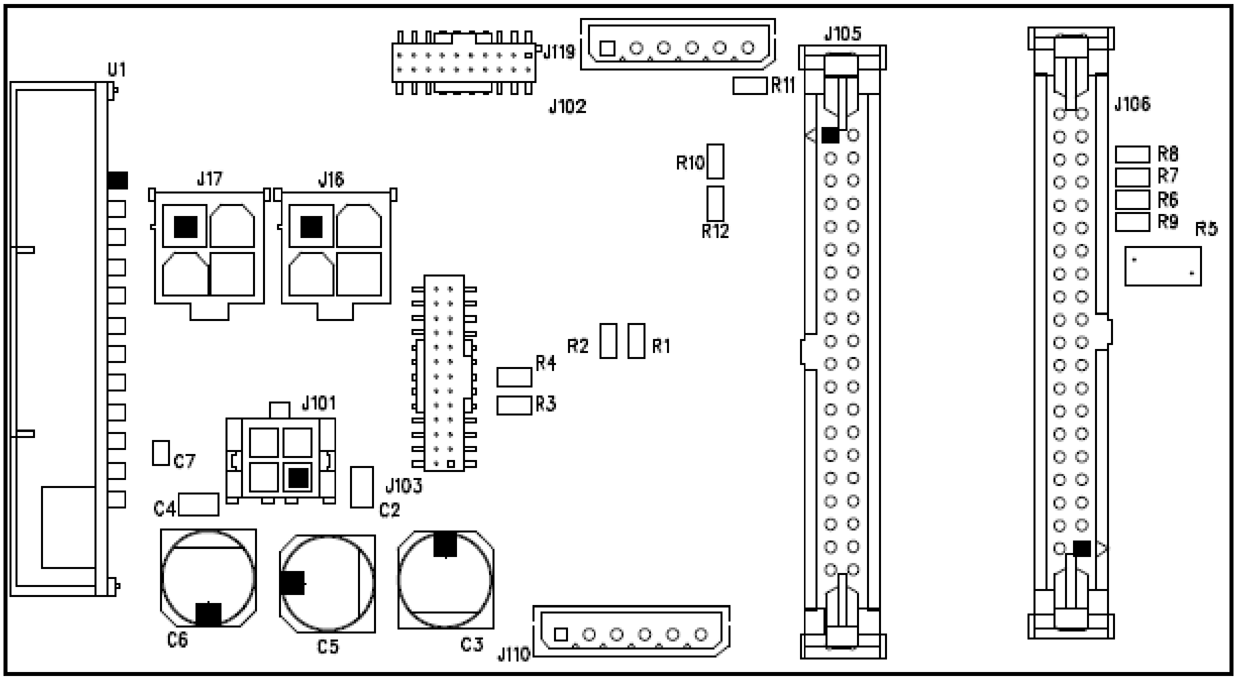


Figure 8 I/O Module Rev F Bottom View

4.5 EC1000 Typical Embedded Installation

The EC1000 and EC1000-IO module were designed to be embedded in laser scanning head or subassembly. The EC1000-IO module fans-out the highly functional I/O of the EC1000 to individual connecters that are more convenient to attach to other equipment such as lasers,encoders, and automation devices.

The EC1000-IO module presents an “inside” interface for attachment to the EC1000 and other devices inside the “head”, and an “outside” interface for attachment to other devices. Typically, other “inside” connections are the laser galvanometer servo controllers that require both DC power and command signal cables. In these situations illustrated by the example diagram in Figure 9, power is provided to the EC1000 through the “outside” power connector on the EC1000-IO module, and is passed through to the EC100 main module and servo boards through connectors on the “inside”. Note that the EC1000-IO module generates the +5 Volts required by the EC1000 module from the +Power input.

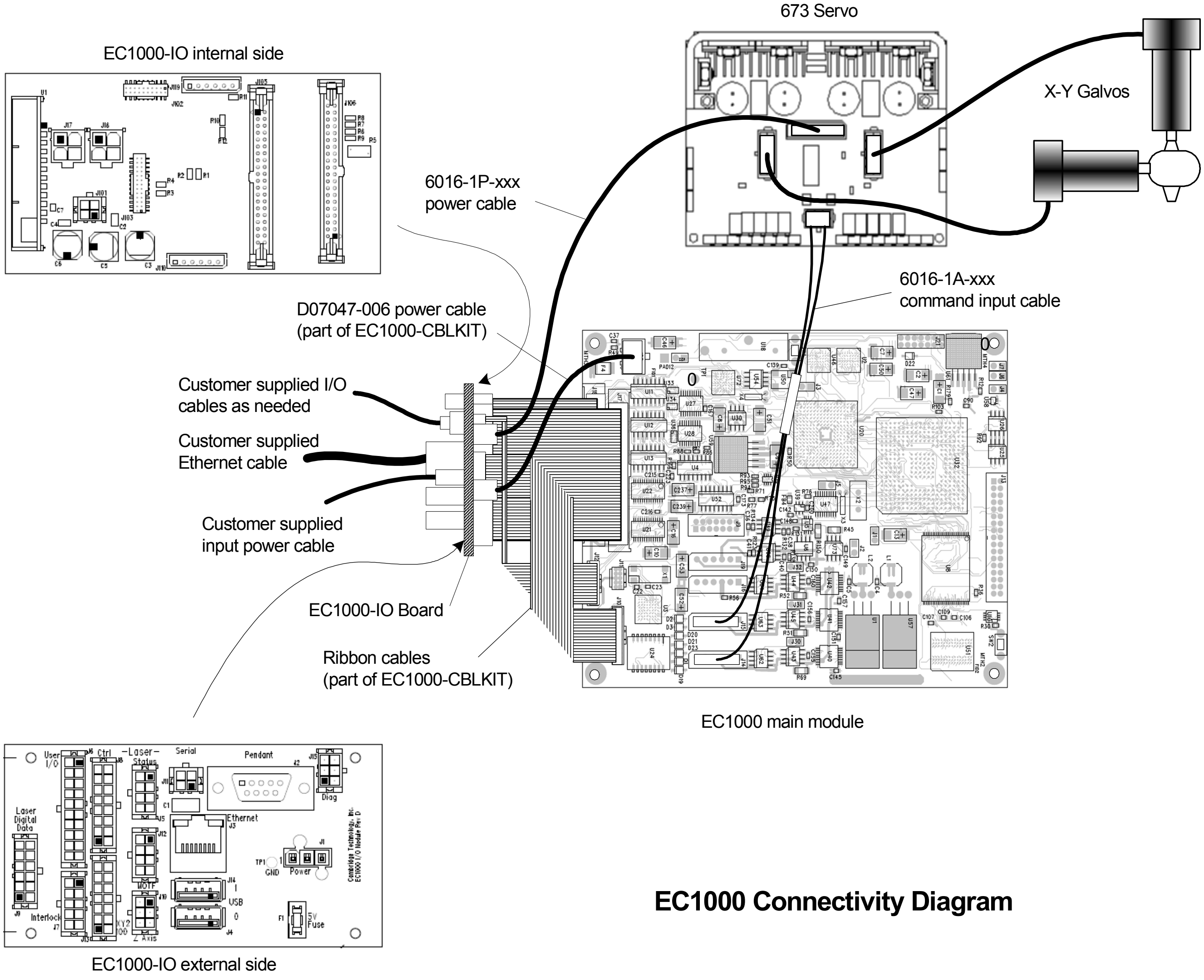


Figure 9 Typical EC1000 Installation

4.6 EC1000 I/O Module Connectivity Summary Reference

The EC1000 I/O Module is provided as a total solution, multi-function board that can be used on both new and old laser heads to provide convenient interface connection. Custom, specialized, and de-populated board variations can be made available for specific uses.

Refer to individual pinout drawings for detailed pinout and signal information.

Table 1: EC1000 to EC1000 I/O Module Connectivity Summary Reference

EC1000 Board Connector	Purpose	EC1000 I/O Board Backside Connector	EC1000 I/O Board User Connector	Purpose
J8	Ethernet & USB	J103	J3	Ethernet
			J4	USB Port 0
			J14	USB Port 1
J12	Serial I/O	J102	J2	Pendant Serial
			J11	Laser Serial Control
			J15	Diagnostics (Debug)
J14	X Servo	N/A - Galvo command connected directly to EC1000		
J15	Y Servo	N/A - Galvo command connected directly to EC1000		
J16	Z Servo	J110	J10	Laser Z Axis Control
J17	Digital I/O	J105	J5	Status
			J8	Laser Control
			J12	Mark on the fly
			J13	XY2-100
J18	Digital I/O	J106	J5	Status
			J6	User Digital I/O
			J7	Interlocks
			J9	Laser Digital Data
J19	Laser Analog Power	J119	J8	Laser Control
J20	Input Power	J101	J1	Input Power
			J16 (back side connector)	X Servo Power
			J17 (back side connector)	Y Servo Power

4.7 EC1000 I/O Module User Connector Part Number Reference

EC1000 I/O Board User Connector	Purpose	Style	Mfg.	Board connector Part #	Mating Connector Part #	Connector Pins
J3	Ethernet	RJ45	SMP	A3002-341-010-101-GY	CAT-5 Ethernet Cable	
J4	USB Port 0	USB-A	REGAL	90S-A-S	USB Type A Device	
J14	USB Port 1	USB-A	REGAL	90S-A-S	USB Type A Device	
J2	Pendant RS232 Serial	DB-9M	AMP	747871-8	DB9 RS-232 DCE	
J101	EC1000 Power	3 mm Microfit, 4 pin	Molex	43045-0414	43025-0400	43030-0009 (20-24GA) 43030-0011 (26-30GA)
J11	Laser RS232 Serial Control	3 mm Microfit, 4 pin	Molex	43045-0414	43025-0400	
J15	Diagnostic RS232 Serial	3 mm Microfit, 6 pin	Molex	43045-0614	43025-0600	
J10	Z Axis Output	3 mm Microfit, 6 pin	Molex	43045-0614	43025-0600	
J8	Laser Digital Control	3 mm Microfit, 16 pin	Molex	43045-1614	43025-1600	
J12	Mark on the fly	3 mm Microfit, 8 pin	Molex	43045-0814	43025-0800	
J13	XY2-100	3 mm Microfit, 14 pin	Molex	43045-1414	43025-1400	
J5	System Status	3 mm Microfit, 8 pin	Molex	43045-0814	43025-0800	
J6	User Digital I/O	3 mm Microfit, 20 pin	Molex	43045-2014	43025-2000	
J7	Interlocks	3 mm Microfit, 10 pin	Molex	43045-1014	43025-1000	
J9	Laser Digital Data	3 mm Microfit, 12 pin	Molex	43045-1214	43025-1200	
J1	Input Power	MiniFit-Jr, 3 pin	Molex	39-30-1039	39-01-4031	39-00-0182
J16 (back side)	X Servo Power	MiniFit-Jr, 4 pin	Molex	39-29-3046	39-01-2045	
J17 (back side)	Y Servo Power	MiniFit-Jr, 4 pin	Molex	39-29-3046	39-01-2045	
J110 (back side)	Z Axis Control from EC1000	2.5MM SPOX	Molex	22-03-5065	50-37-5063	08-70-1040
J119 (back side)	Laser Analog from EC1000	2.5MM SPOX	Molex	22-03-5065	50-37-5063	

4.8 EC1000 to I/O Module Connector Part Number Reference

EC1000 Board Connector	Purpose	EC1000 I/O Board Backside Connector	Style	Mfg.	Board Connector Part #	Mating Cable or Connector/Pins
J8	Ethernet & USB	J103	0.050 X 2 X 13	Samtec	FTSH-113-01-LM-D-K FTSH-113-01-LM-DV-K	FFSD-13-D-06.00-01-N
J12	Serial I/O	J102	0.050 X 2 X 10	Samtec	FTSH-110-01-LM-D-K FTSH-110-01-LM-DV-K	FFSD-10-D-06.00-01-N
J17	User I/O "A"	J105	2mm X 2 X 20	Samtec	EHT-120-01-S-D STMM-120-02-SM-D	TCSD-20-D-03.00-01-N
J18	User I/O "B"	J106	2mm X 2 X 20	Samtec	EHT-120-01-S-D STMM-120-02-SM-D	TCSD-20-D-02.00-01-N
J16	Z Servo	J110	2.5mm X 1 X 6	Molex	22-03-5065	50-37-5063/08-70-1040
J19	Laser Analog	J119	2.5mm X 1 X 6	Molex	22-03-5065	50-37-5063/08-70-1040
J20	DC Input Power	J101	3 mm Microfit X 4	Molex	43045-0414	43025-0400/43030

4.9 EC1000 Servo Controller Connector Part Number Reference

EC1000 Board Connector	Purpose	Style	Mfg.	Board Connector Part #	Mating Cable or Connector/Pins
J14, J15, J16	X, Y, Z Servo	2.5mm X 1 X 6	Molex	22-03-5065	50-37-5063/08-70-1040
J19	Laser Analog Out	2.5mm X 1 X 6	Molex	22-03-5065	50-37-5063/08-70-1040

4.10 EC1000 Signal Conditioning

Most control connections are optically-isolated on the EC1000 main module. On the EC1000-IO module some signal sets are further pre-conditioned to simplify system wiring when optical ioslation is not important. To handle these cases the EC1000-IO module connectors are wired so that simple connector jumpers can be used to connect EC1000 system power and ground for satisfyng the iosolator drive requirements. The following figures illustrate the input and output optical isolation used with the various signal groups.

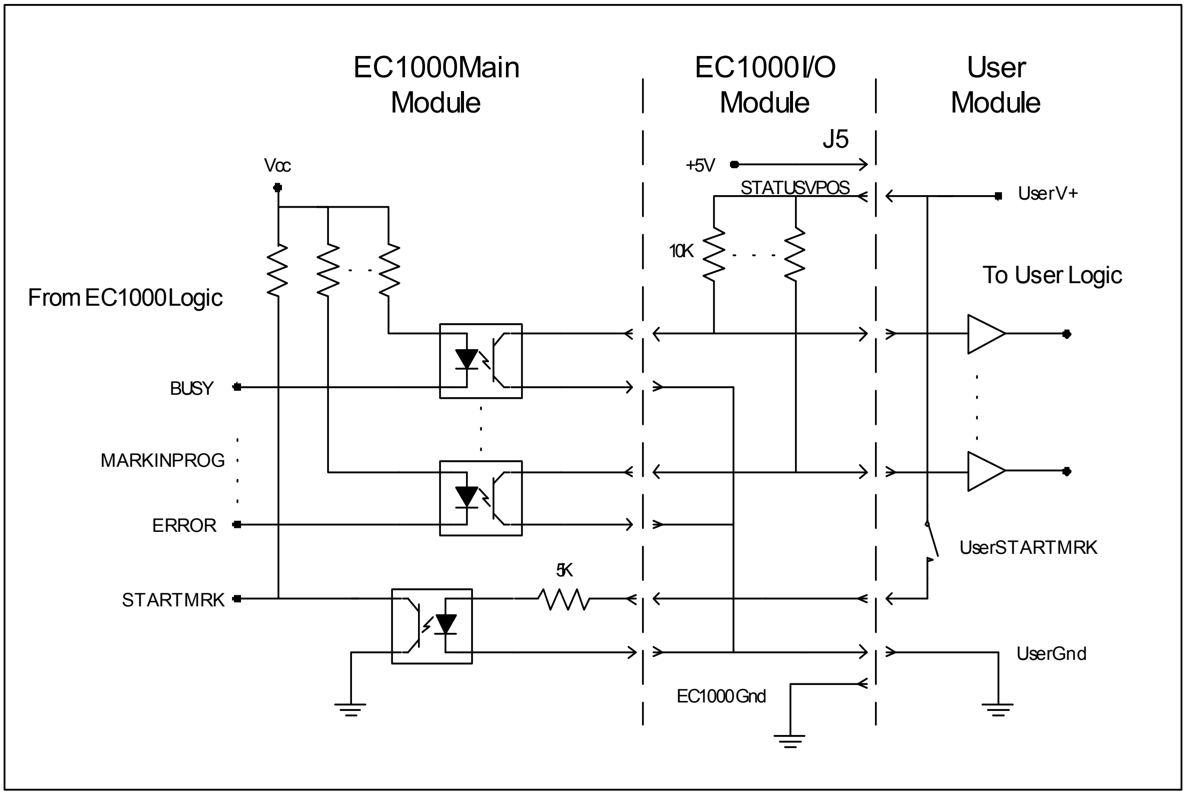


Figure 10 EC1000 System Control and Status Optical Isolation

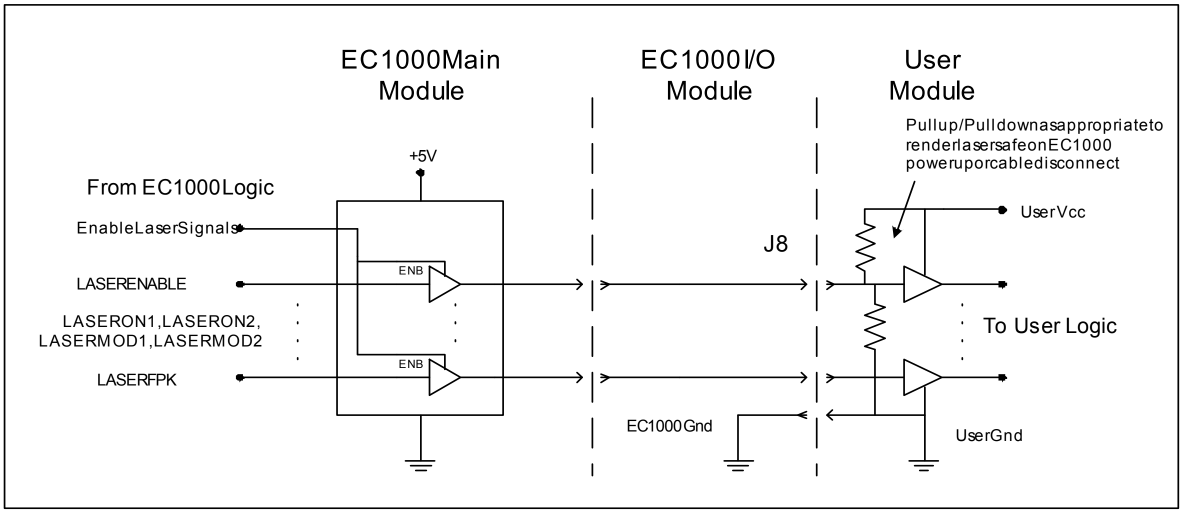


Figure 11 EC1000 Laser Control Signal Conditioning

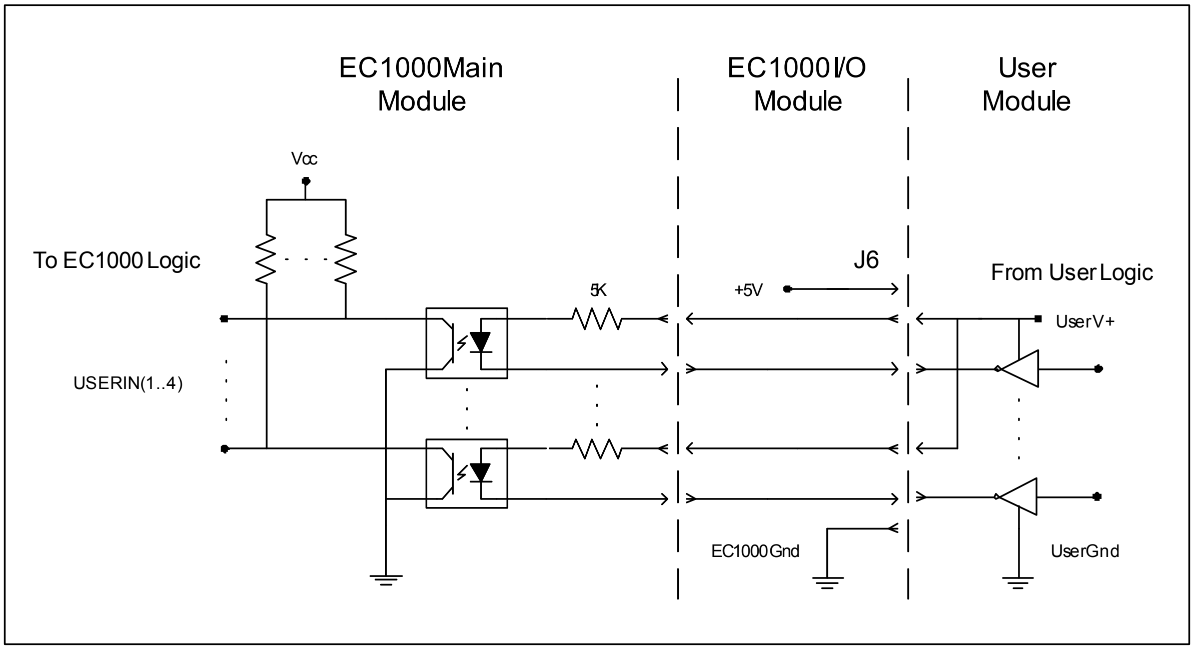


Figure 12 EC1000 User Inputs Optical Isolation

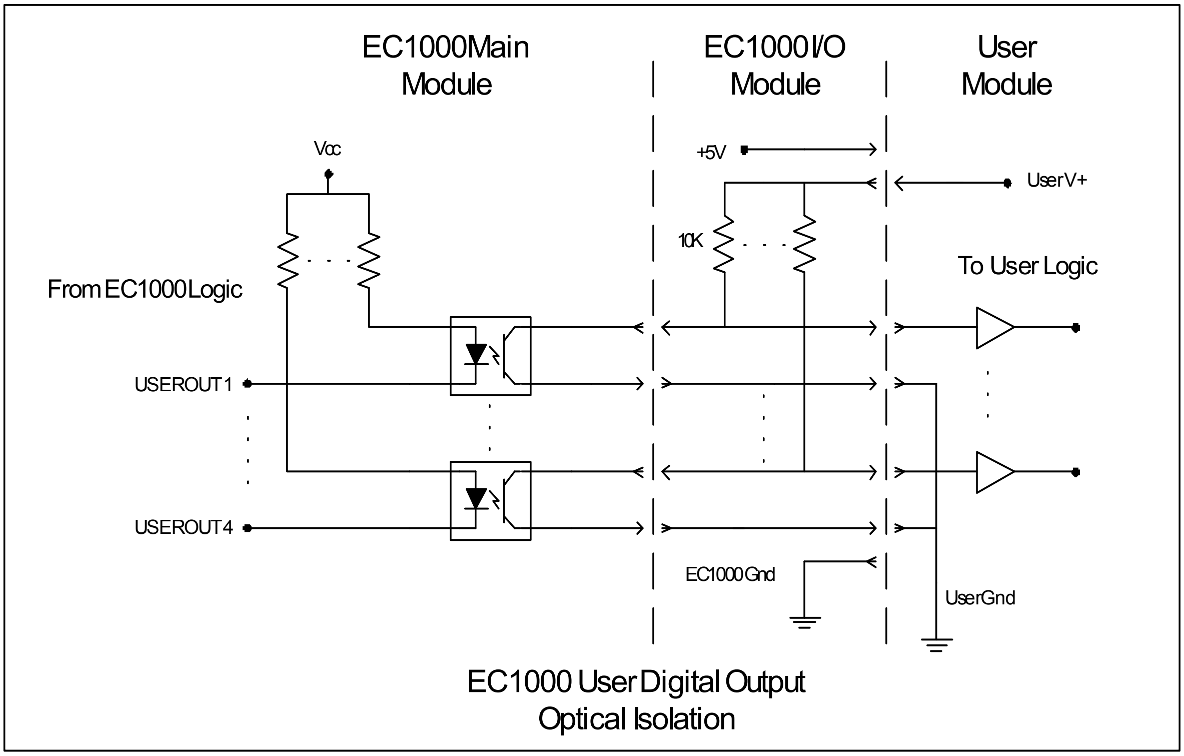


Figure 13 EC1000 User Outputs Optical Isolation

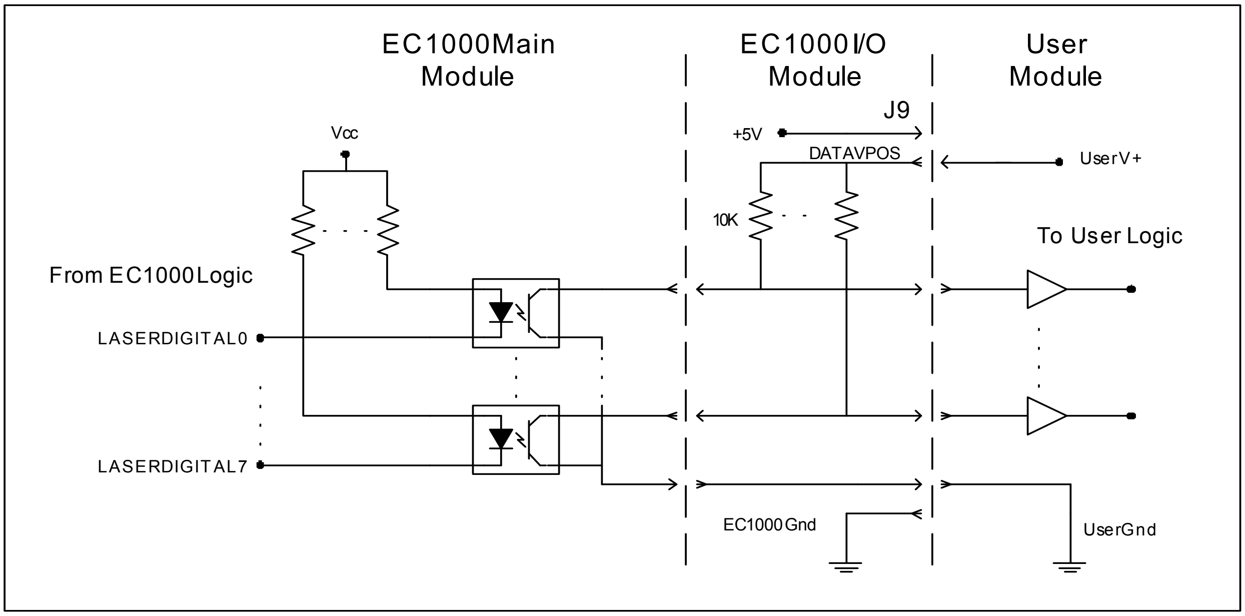


Figure 14 EC1000 Laser Digital Output Optical Isolation

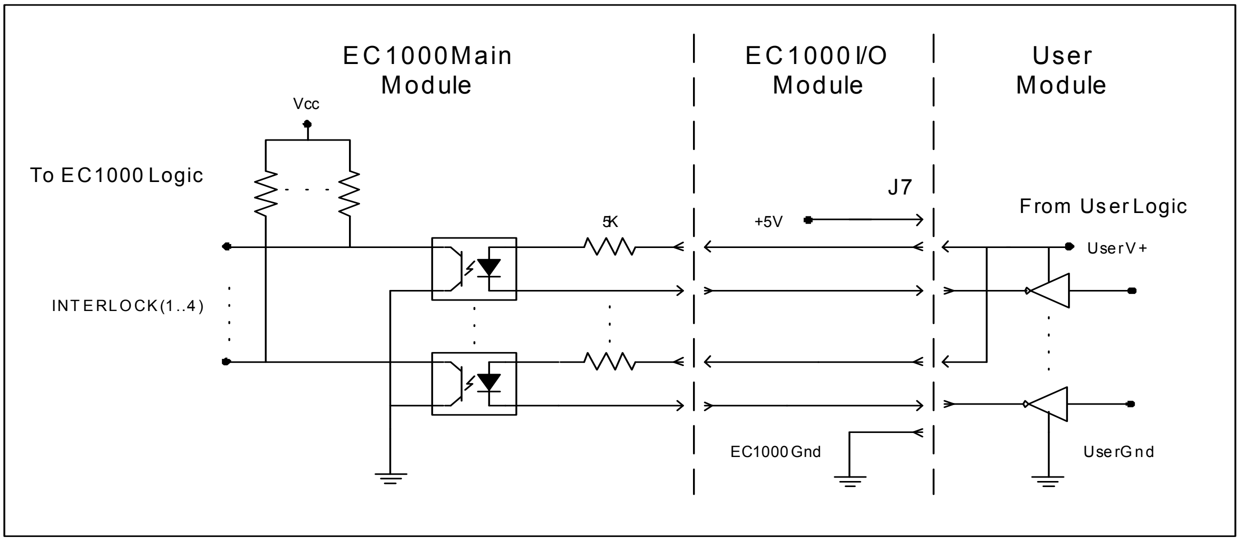


Figure 15 EC1000 System Interlock Optical Isolation

4.11 EC1000 and EC1000 I/O Module Connectors

This section contains pinout drawings and connectivity details for EC1000 Board connectors to the EC1000 I/O Module.

4.11.1 Ethernet/USB Connectors

Connector J8 on the EC1000 board provides access to Ethernet and USB connectivity. The figure below details pinouts and relevant pinouts and signal names for the EC1000 I/O Module.

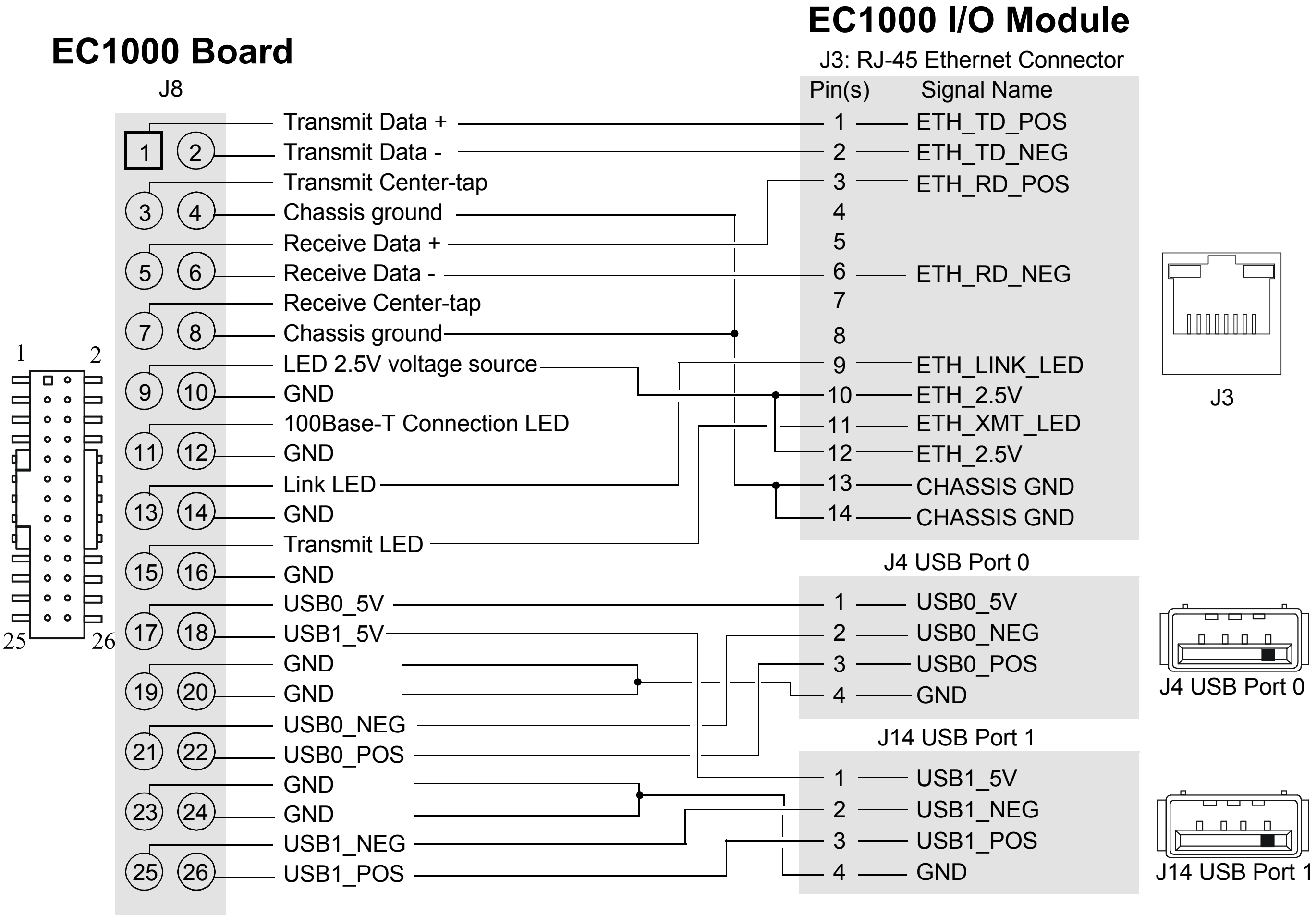
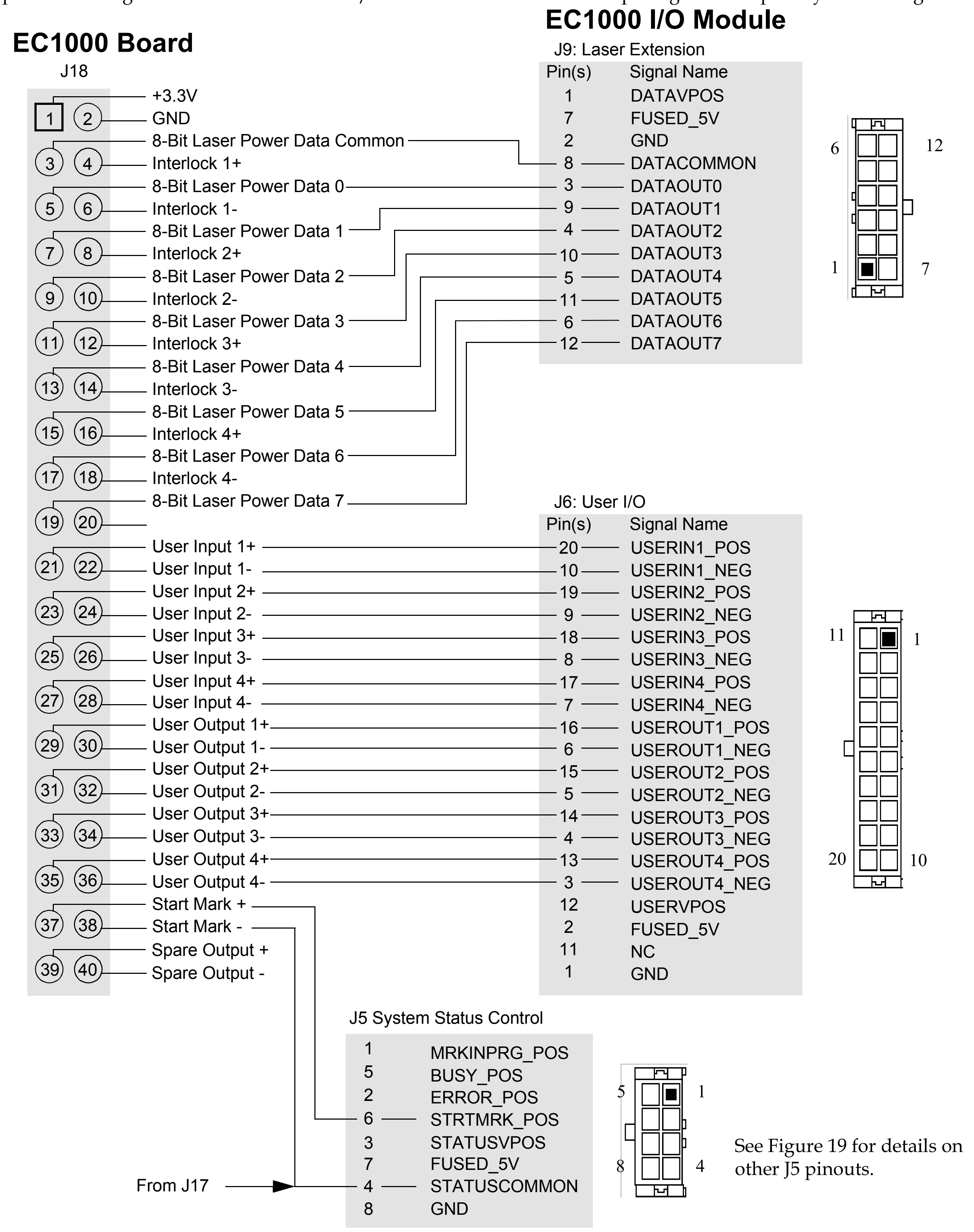


Figure 16 EC1000 Ethernet/USB Connector Pinouts and Connectivity to EC1000 I/O Module

4.11.2 USER I/O A Connector

Connector J18 on the EC1000 board provides access to User I/O. The figures below details pinouts and relevant pinouts and signal names for the EC1000 I/O Module connectors. All output signals are optically isolated signals.



J9: Laser Extension

Pin(s)	Signal Name
1	DATAVPOS
7	FUSED_5V
2	GND
8	DATACOMMON
3	DATAOUT0
9	DATAOUT1
4	DATAOUT2
10	DATAOUT3
5	DATAOUT4
11	DATAOUT5
6	DATAOUT6
12	DATAOUT7

J6: User I/O

Pin(s)	Signal Name
20	USERIN1_POS
10	USERIN1_NEG
19	USERIN2_POS
9	USERIN2_NEG
18	USERIN3_POS
8	USERIN3_NEG
17	USERIN4_POS
7	USERIN4_NEG
16	USEROUT1_POS
6	USEROUT1_NEG
15	USEROUT2_POS
5	USEROUT2_NEG
14	USEROUT3_POS
4	USEROUT3_NEG
13	USEROUT4_POS
3	USEROUT4_NEG
12	USERVPOS
2	FUSED_5V
11	NC
1	GND

J5 System Status Control

1	MRKINPRG_POS
5	BUSY_POS
2	ERROR_POS
6	STRTMRK_POS
3	STATUSVPOS
7	FUSED_5V
4	STATUSCOMMON
8	GND

See Figure 19 for details on other J5 pinouts.

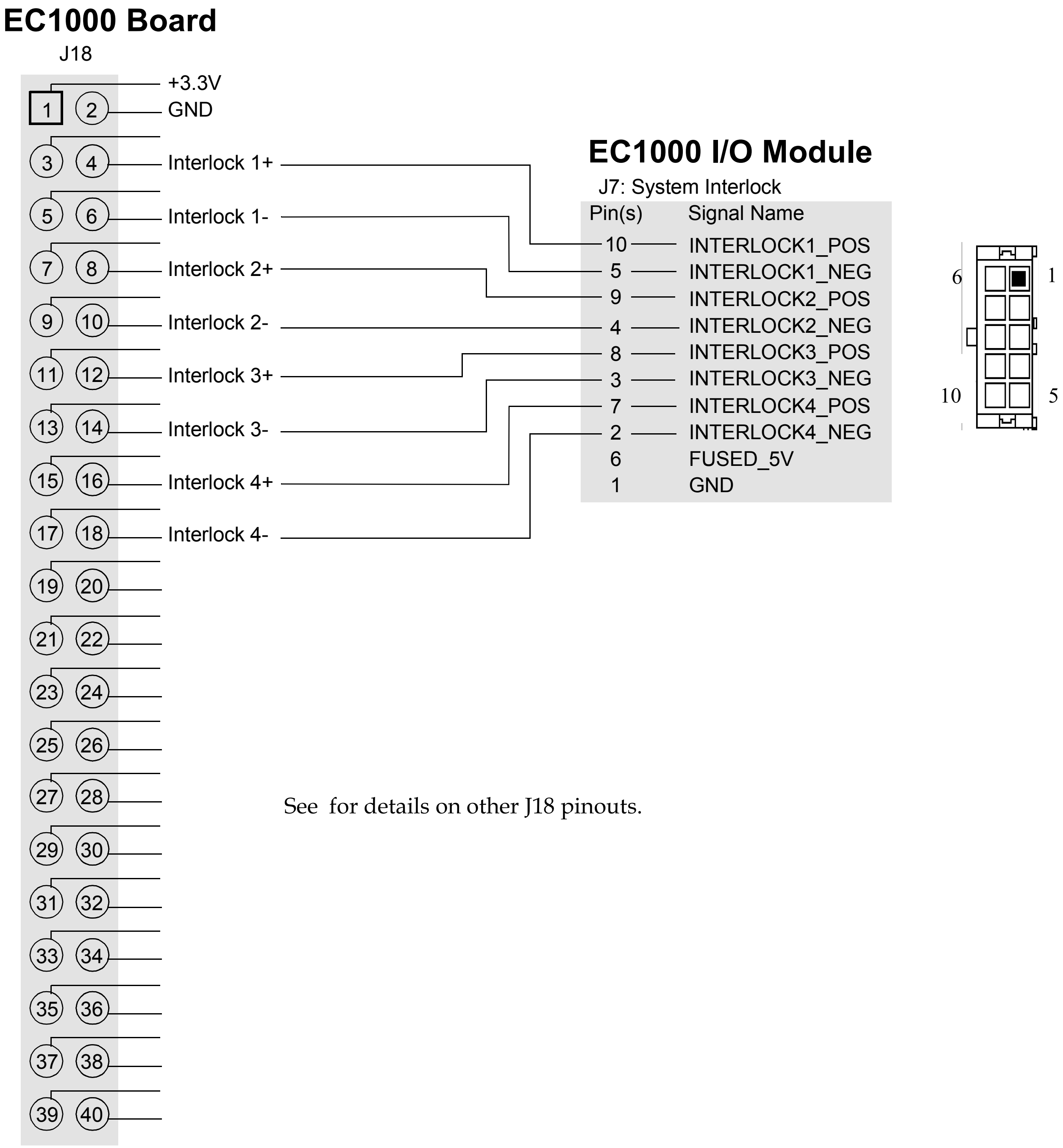
Figure 17 EC1000 J18 User I/O Pinouts and Connectivity to EC1000 I/O Module connectors, 1 of 2

The 8-bit laser power data connectors are intended to provide digital representation of the laser power information.

The 0-255 signal range corresponds to a voltage range of 0-10V for control of a digital laser. (Direct analog control of a laser is provided through connector J8.) Four interlock lines are provided to provide interlock protection to the scanning system, which can be connected to safety switches (for example, panel switches). All laser pattern generation will stop immediately, any current job will be aborted and will need to be reset if an interlock is broken

There are four user inputs that are intended to provide external synchronization capability with external equipment (for example, a loader or handler). The four independent bits can be used to pause a job.

There are four independent user outputs that are intended to provide system status/state information to external equipment and/or user signalling devices. A fifth input, the Start Mark signal, is provided for backward compatiblity with systems that may require it, but the signal can be considered a fifth user input and can be connected the J5 connector.



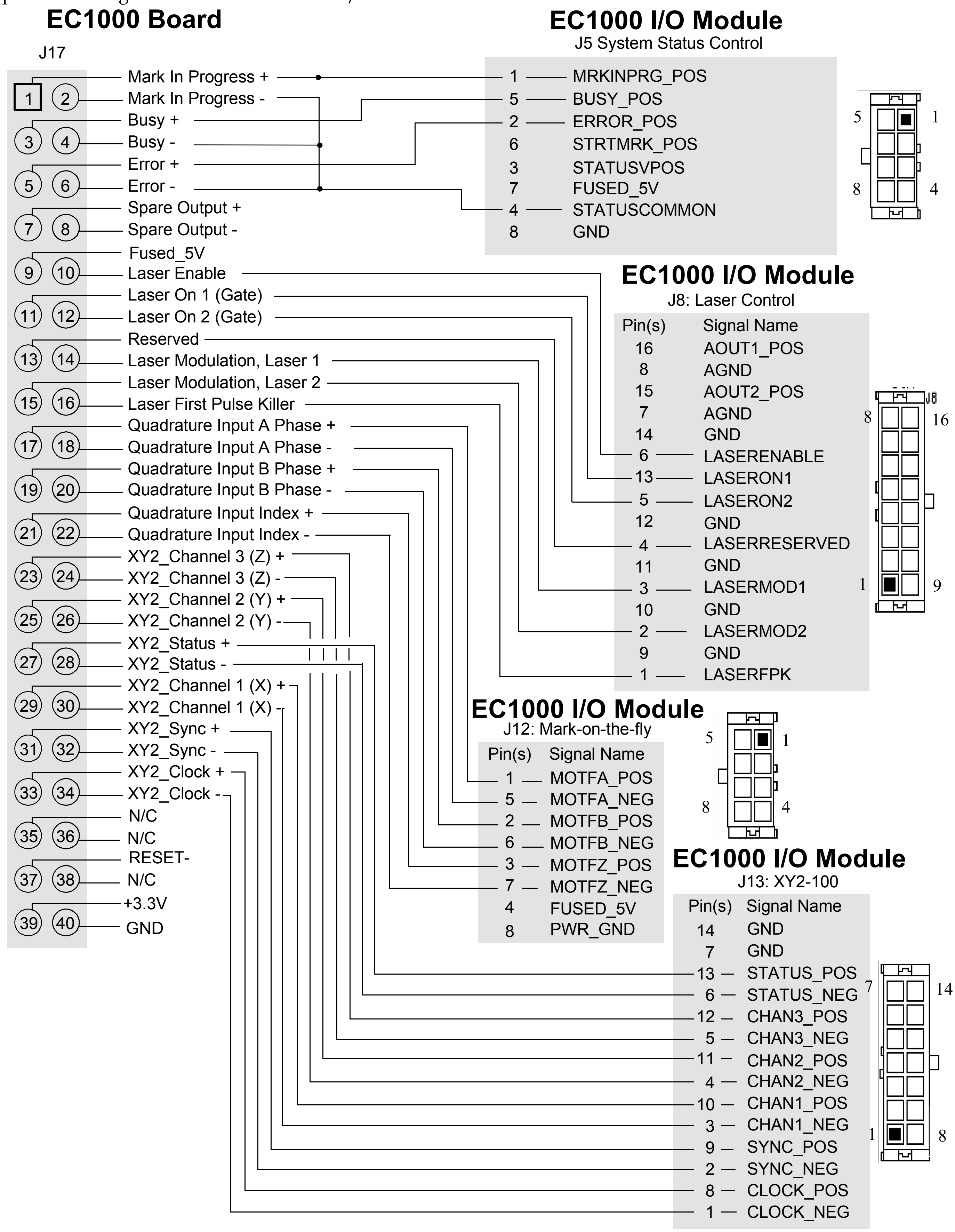
Pin(s)	Signal Name
10	INTERLOCK1_POS
5	INTERLOCK1_NEG
9	INTERLOCK2_POS
4	INTERLOCK2_NEG
8	INTERLOCK3_POS
3	INTERLOCK3_NEG
7	INTERLOCK4_POS
2	INTERLOCK4_NEG
6	FUSED_5V
1	GND

See for details on other J18 pinouts.

Figure 18 EC1000 J18 User I/O Pinouts and Connectivity to EC1000 I/O Module connectors, 2 of 2

4.11.3 USER I/O B Connector

Connector J17 on the EC1000 board provides access to User I/O. The figures below details pinouts and relevant pinouts and signal names for the EC1000 I/O Module connectors.



EC1000 I/O Module — J5 System Status Control

1	MRKINPRG_POS
5	BUSY_POS
2	ERROR_POS
6	STRTMRK_POS
3	STATUSVPOS
7	FUSED_5V
4	STATUSCOMMON
8	GND

EC1000 I/O Module — J8: Laser Control

Pin(s)	Signal Name
16	AOUT1_POS
8	AGND
15	AOUT2_POS
7	AGND
14	GND
6	LASERENABLE
13	LASERON1
5	LASERON2
12	GND
4	LASERRESERVED
11	GND
3	LASERMOD1
10	GND
2	LASERMOD2
9	GND
1	LASERFPK

EC1000 I/O Module — J12: Mark-on-the-fly

Pin(s)	Signal Name
1	MOTFA_POS
5	MOTFA_NEG
2	MOTFB_POS
6	MOTFB_NEG
3	MOTFZ_POS
7	MOTFZ_NEG
4	FUSED_5V
8	PWR_GND

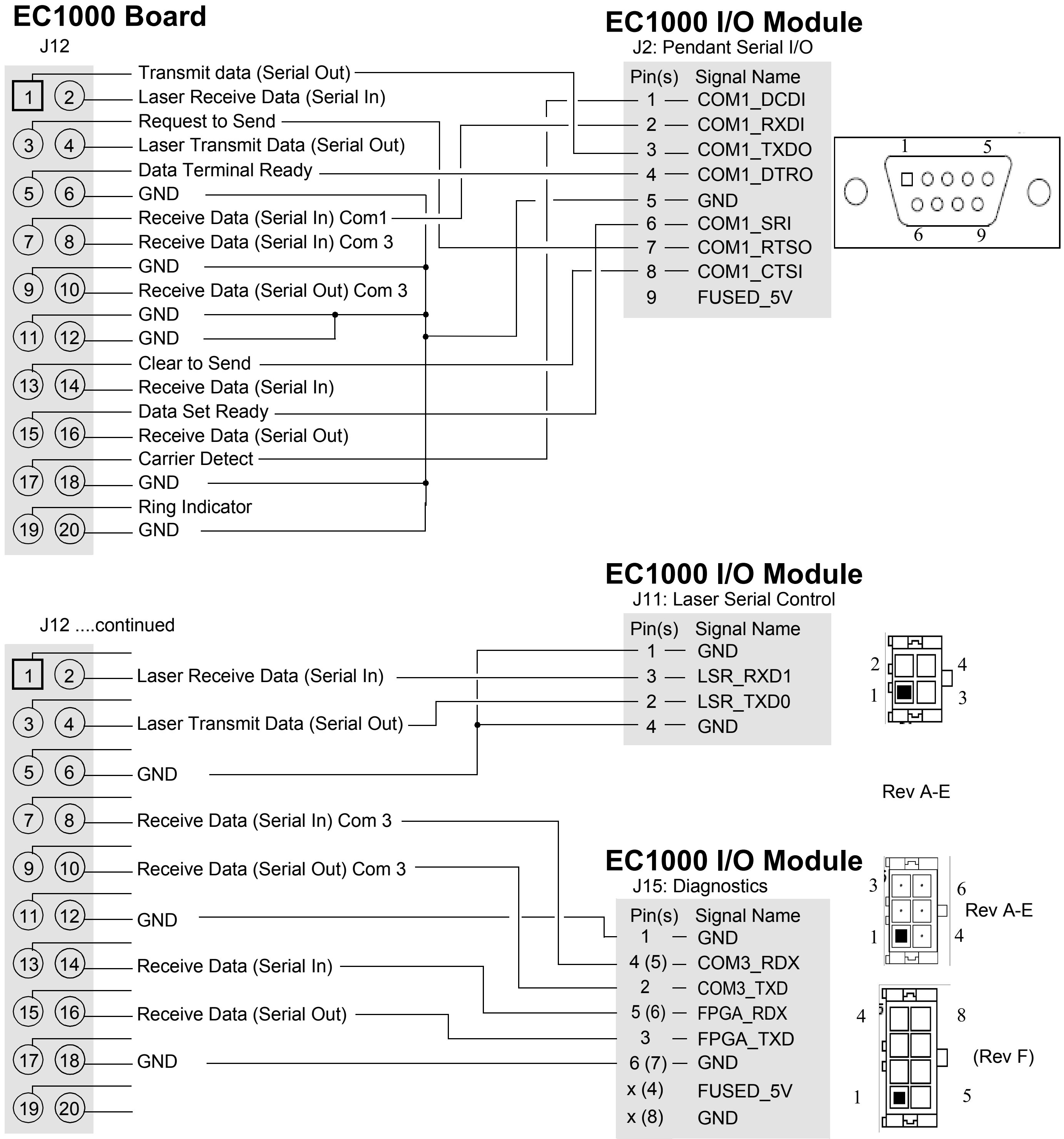
EC1000 I/O Module — J13: XY2-100

Pin(s)	Signal Name
14	GND
7	GND
13	STATUS_POS
6	STATUS_NEG
12	CHAN3_POS
5	CHAN3_NEG
11	CHAN2_POS
4	CHAN2_NEG
10	CHAN1_POS
3	CHAN1_NEG
9	SYNC_POS
2	SYNC_NEG
8	CLOCK_POS
1	CLOCK_NEG

Figure 19 EC1000 J17 User I/O Pinouts and Connectivity to EC1000 I/O Module connectors

4.11.4 Low Speed Serial Connectors

Connector J12 on the EC1000 board provides access to Low Speed Serial connection. The figures below details pinouts and relevant pinouts and signal names for the EC1000 I/O Module connectors.


EC1000 Board
J12
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Transmit data (Serial Out)
Laser Receive Data (Serial In)
Request to Send
Laser Transmit Data (Serial Out)
Data Terminal Ready
GND
Receive Data (Serial In) Com1
Receive Data (Serial In) Com 3
GND
Receive Data (Serial Out) Com 3
GND
GND
Clear to Send
Receive Data (Serial In)
Data Set Ready
Receive Data (Serial Out)
Carrier Detect
GND
Ring Indicator
GND
EC1000 I/O Module
J2: Pendant Serial I/O
1
5
6
9
J12 ....continued
Laser Receive Data (Serial In)
Laser Transmit Data (Serial Out)
GND
Receive Data (Serial In) Com 3
Receive Data (Serial Out) Com 3
GND
Receive Data (Serial In)
Receive Data (Serial Out)
GND
EC1000 I/O Module
J11: Laser Serial Control
2
4
1
3
Rev A-E
EC1000 I/O Module
J15: Diagnostics
3
6
1
4
Rev A-E
4
8
1
5
(Rev F)

EC1000 I/O Module — J2: Pendant Serial I/O

Pin(s)	Signal Name
1	COM1_DCDI
2	COM1_RXDI
3	COM1_TXDO
4	COM1_DTRO
5	GND
6	COM1_SRI
7	COM1_RTSO
8	COM1_CTSI
9	FUSED_5V

EC1000 I/O Module — J11: Laser Serial Control

Pin(s)	Signal Name
1	GND
3	LSR_RXD1
2	LSR_TXD0
4	GND

EC1000 I/O Module — J15: Diagnostics

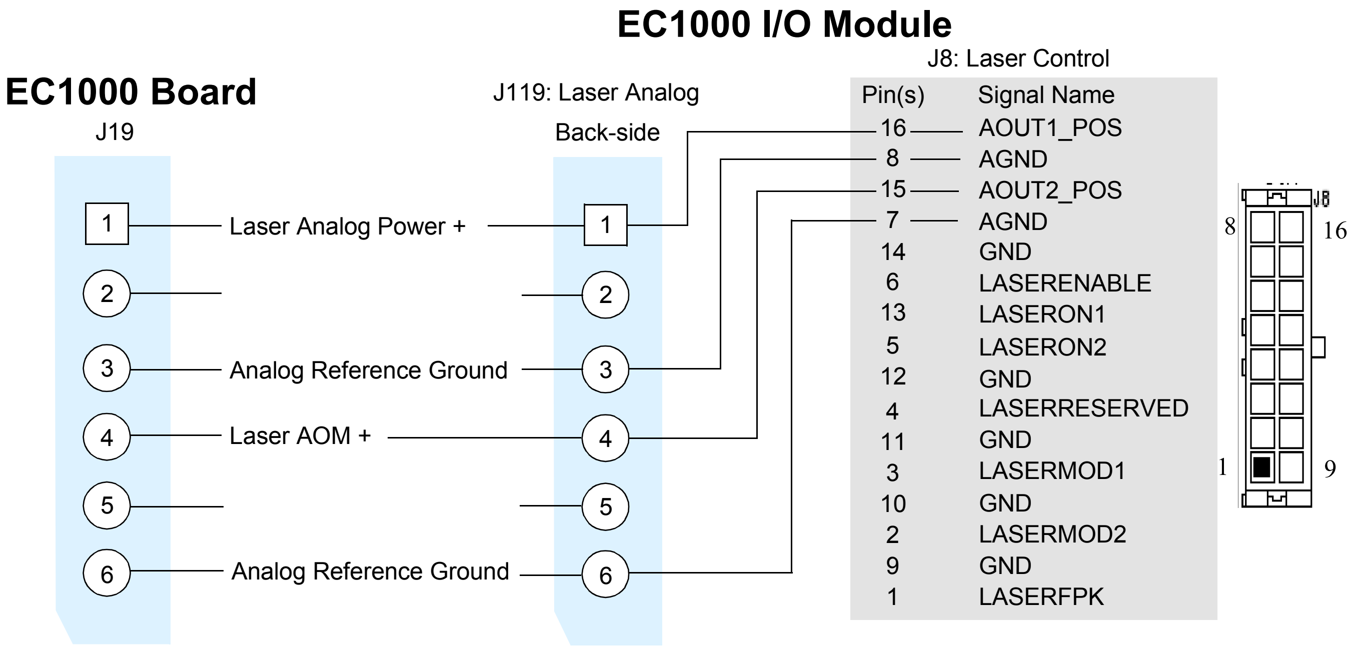
Pin(s)	Signal Name
1	GND
4 (5)	COM3_RDX
2	COM3_TXD
5 (6)	FPGA_RDX
3	FPGA_TXD
6 (7)	GND
x (4)	FUSED_5V
x (8)	GND

Figure 20 EC1000 J12 Low Speed Serial Pinouts and Connectivity to EC1000 I/O Module connectors

4.11.5 Laser Analog Connector

Connector J19 on the EC1000 board provides access to Laser Analog connections. The figures below details pinouts and relevant pinouts and signal names for the EC1000 I/O Module connectors.

The Laser Analog Power connectors are differential analog signals that go from 0-10V (not negative). The intent of the laser analog power is to set the laser power level for the duration of a job. The AOM is a setting that can change very rapidly, for example, to modulate the signal for raster bit-map type patterning, to vary each pixel's intensity.



Pin(s)	Signal Name
16	AOUT1_POS
8	AGND
15	AOUT2_POS
7	AGND
14	GND
6	LASERENABLE
13	LASERON1
5	LASERON2
12	GND
4	LASERRESERVED
11	GND
3	LASERMOD1
10	GND
2	LASERMOD2
9	GND
1	LASERFPK

Figure 21 EC1000 J19 Laser Analog Pinouts and Connectivity to EC1000 I/O Module connectors

4.11.6 XYZ Analog Signal Descriptions

The position command outputs for the X, Y and Z axes are differential analog signals that are both actively driven. If it's a single-ended application, the reference is the analog ground reference and the positive signal (plus signal) will be the controlling output signal from the control board.

With differential analog signals, the signals are in relation to one another (+ and -), for example if the plus signal = 2 and the negative signal = -2 the differential is 4. This method is used on some controllers to reduce and/or compensate for inherent signal noise, particulary when transfering signals over long distances.

⚠ **CAUTION** ⚠

Never connect the negative (minus) signal on the board to ground or you may damage the board. The minus signal is actively driven by a driver op-amp.

4.11.7 X & Y-Axis Connectors

Connectors J14 & J15 on the EC1000 board provides access to X & Y-Axis connections, respectively. The figure below details the connector pinouts. Since the X & Y-Axis servo control electronics are expected to be contained within a marking head along with the EC1000 controller, there are no provisions to provide external access to these signals via the I/O Module.

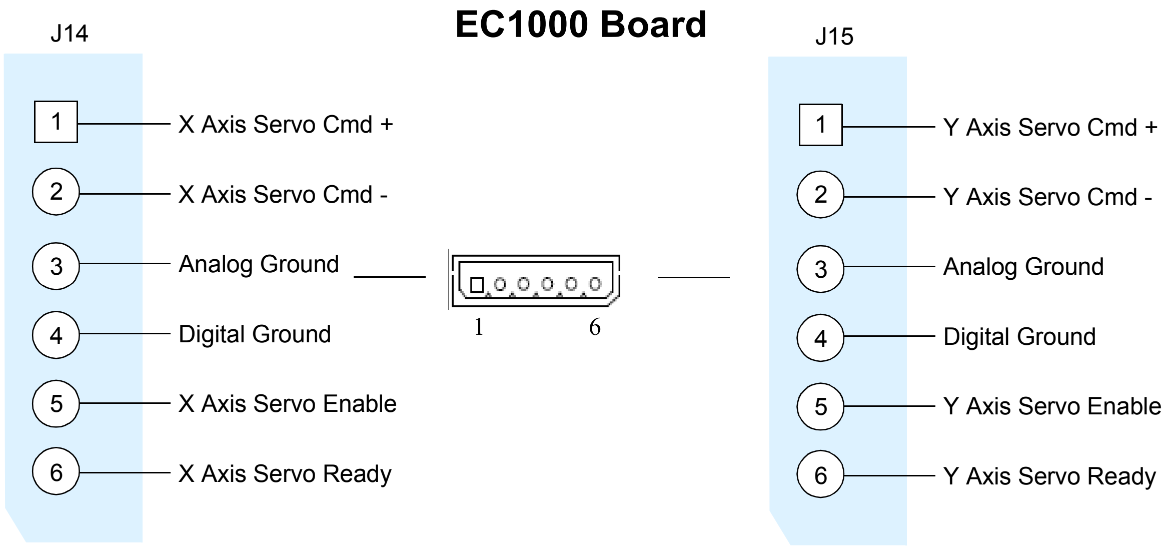


Figure 22 EC1000 J14 & J15, X & Y-Axis Servo Controller Pinouts

4.11.8 Z-Axis Connectors

The Z axis is optionally connected to the I/O Module, if a laser head has Z-axis (focus) control capability.

Connector J16 on the EC1000 board provides access to Z-Axis connections. The figures below details pinouts and relevant pinouts and signal names for the EC1000 I/O Module connectors. The Z-Axis signals are carried over to the I/O Module to provide external access to focusing systems that are not contained inside the marking head.

Alternatively, the signals may terminate directly at X-Axis servo control electronics within the head.

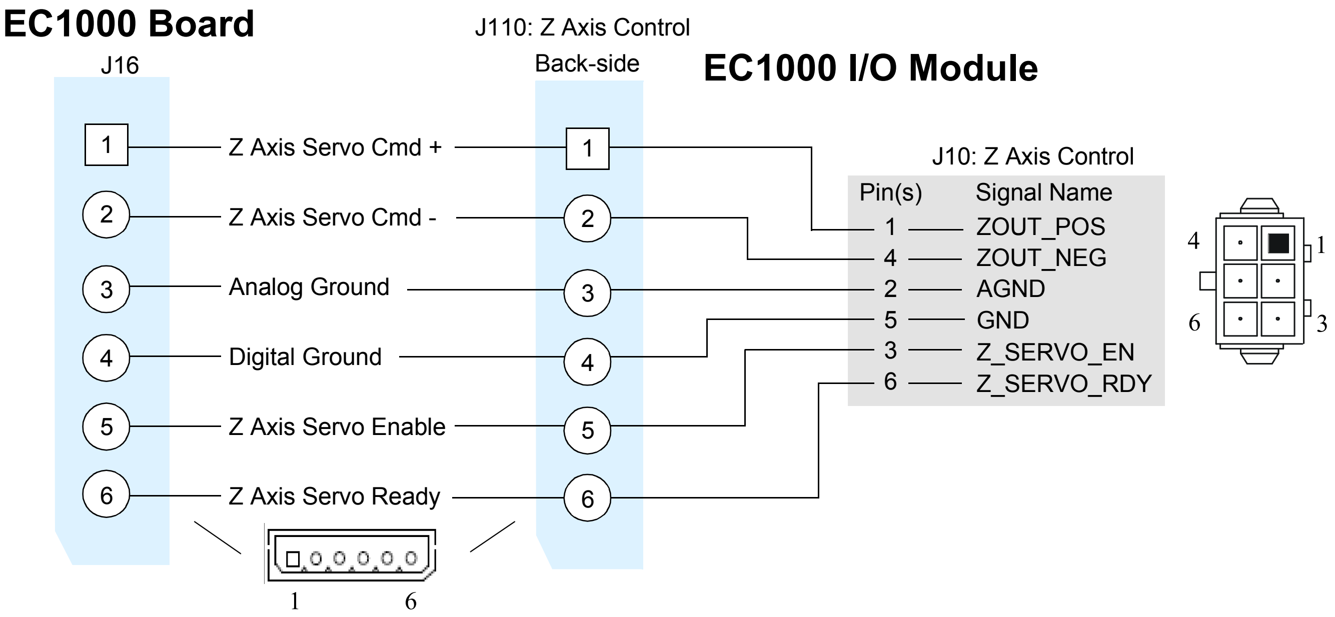
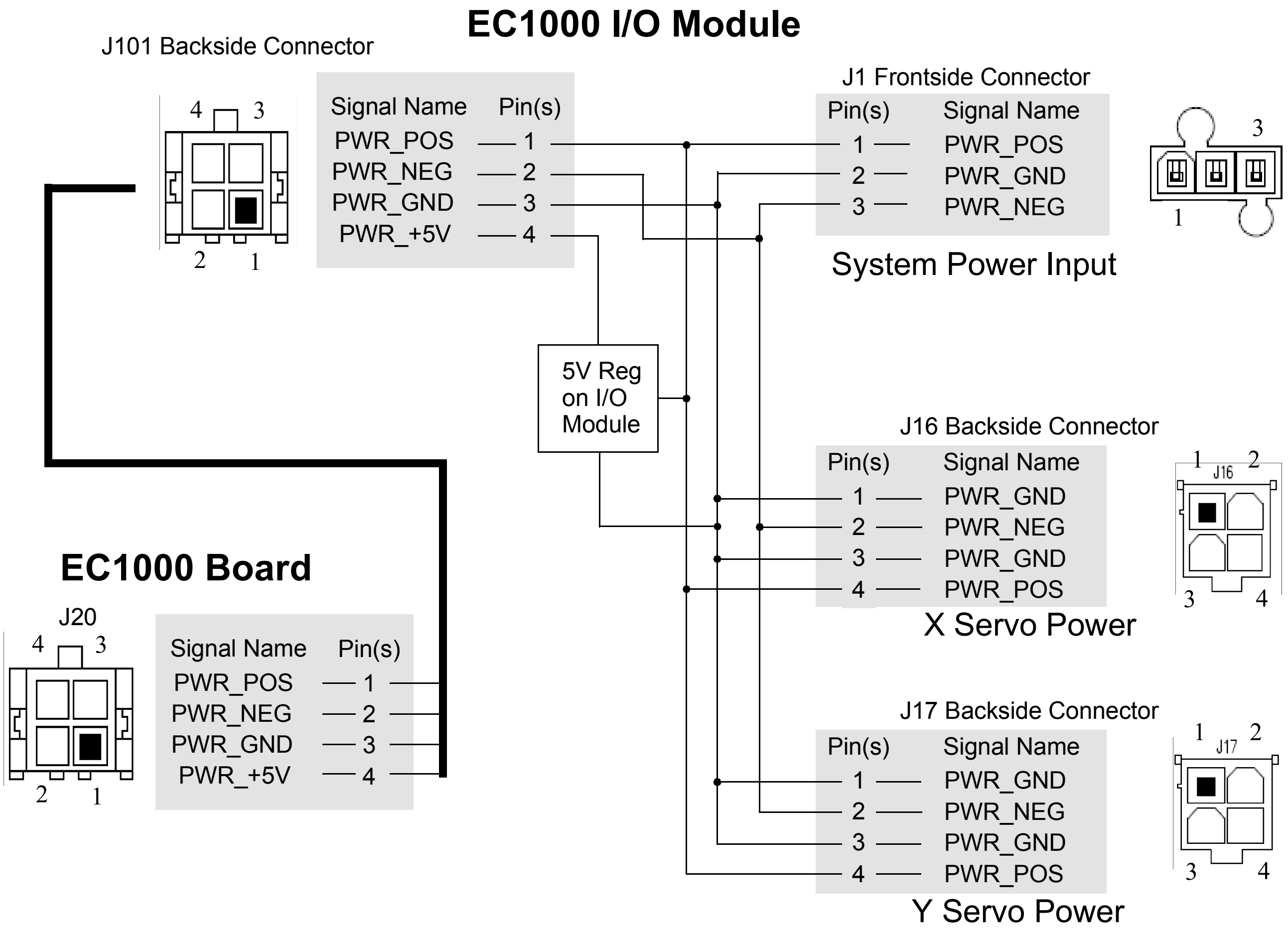


Figure 23 EC1000 J16 Z-Axis Pinouts and Connectivity to EC1000 I/O Module connector

4.11.9 Power Connectors

J1 on the EC1000 I/O module is the main input power connector when both the E1000 and EC1000-IO modules are used together. The IO module redistributes the power to three connectors: J16 and J17, located on the back-side are intended to provide power to the X & Y galvo servo drivers. J101, also located on the back-side, provided power to the EC1000 main module. The I/O module contains a +5Vswitching regulator that creates +5V from the + power suply and delivers it to the EC1000 power connector, J101.



J101 Backside Connector

Signal Name	Pin(s)
PWR_POS	1
PWR_NEG	2
PWR_GND	3
PWR_+5V	4

J1 Frontside Connector

Pin(s)	Signal Name
1	PWR_POS
2	PWR_GND
3	PWR_NEG

J16 Backside Connector

Pin(s)	Signal Name
1	PWR_GND
2	PWR_NEG
3	PWR_GND
4	PWR_POS

EC1000 Board J20

Signal Name	Pin(s)
PWR_POS	1
PWR_NEG	2
PWR_GND	3
PWR_+5V	4

J17 Backside Connector

Pin(s)	Signal Name
1	PWR_GND
2	PWR_NEG
3	PWR_GND
4	PWR_POS

Figure 24   EC1000 J20 Power Pinouts and Connectivity to EC1000 I/O Module Connector (backside shown also)

4.12 XY2-100 Protocol Interface

The XY2-100 Protocol is a serial digital protocol that allows you to connect the controller to a scan head, and send command data digitally. Each command is broken up and sent in a 20-bit serial data stream. The stream is received at the scan head (and re-assembled into a parallel "word") and converted via an A/D (analog/digital) converter to an analog signal to drive the head galvonometers.

The signals that comprise the XY2-100 protocol are accessed via the EC1000-IO module at connector J13.

4.12.1 XY2-100 Interface Timing

Figure 25 shows the timing of the XY2-100 interface.

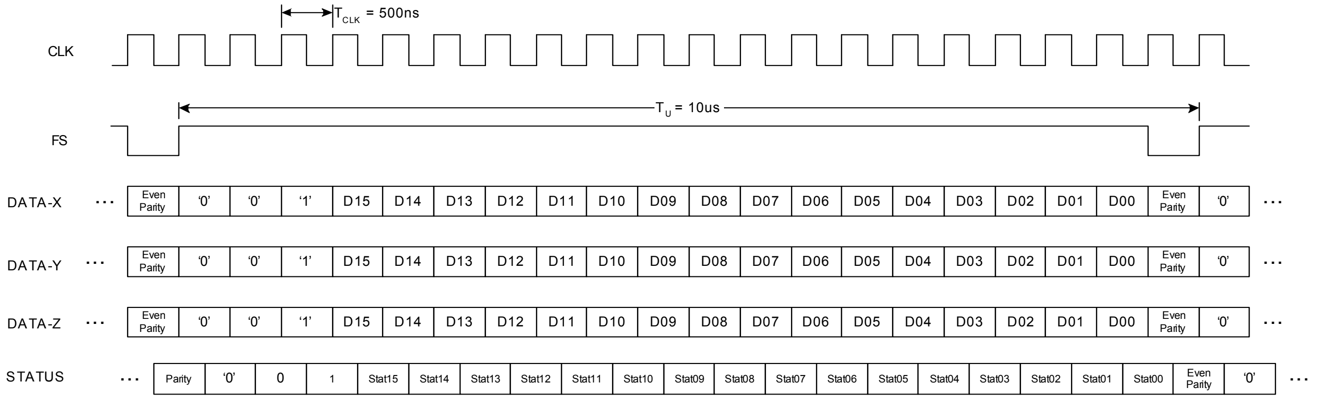


Figure 25 XY2-100 Interface Timing

The EC1000 generates a digital serial stream for each of the axes and transmits them out the RS-422 tranceivers located on the main module. These signals are routed to connector J13 on the EC1000-I/O module. The serial status stream is de-serialized by the EC1000 and the resulting 16-bit value is presented in the XY2-100 Status register for application use. The register is accessed by sending a register read request to the EC1000. See section: Priority Data Definition

4.13 Stand-alone Operation

The EC1000 can function as a stand-alone controller providing laser system control without the need for a host computer attached. In stand-alone mode, user interaction with the controller for job selection, job adjustment, and administrative maintenance is provided through the use of an attached pendant. The EC1000 supports QSI Corporation's QTERM-J10 terminal attached through a standard RS-232 9-pin "D" interface connector (J2) located on the EC1000-I/O board. This connector provides +5V on pin 9 for powering the QTERM-J10 module in anticipation of using a QTERM-J10 with the +5V without power regulator option.

Standalone operation is enabled by appropriate interactions with the pendant, or by accessing the EC1000 via remote control (section 7.2 Protocol Specification).

5 Principle Of Operation

The EC1000 controls a laser system's galvanometers, accurately positioning deflection mirrors in synchronization with laser control signals. The sequence of motions, the speed of operation, the power that the laser uses, and the synchronization with external equipment is expressed in scanning jobs. These jobs consist of sequences of instructions to the marking engine located on the EC1000 module. Some instructions configure the module such as setting up to emit laser control signals with the appropriate timing relative to the commanded motion of the laser steering galvos. The bulk of the instructions, however, are sequences of *mark* and *jump* instructions, which describe when and where to move the galvos and when to gate the laser control signals relative to those motions.

Job data is typically prepared using editor applications designed for that purpose. These applications may be custom software applications written by an OEM integrator, or one of several commercially available packages. These applications are hosted on a Microsoft Windows^{TM} based PC and interface to the EC1000 modules through a Microsoft COM DLL library. The library takes care of establishing and maintaining communications with an EC1000, and provides a managed conduit for passing data to and from the controller. Figure 26 illustrates this arrangement.

The following sections describe the hardware and software architecture of the EC1000 system and define the COM DLL Application Programming Interface (API) that would be used to control the EC1000.

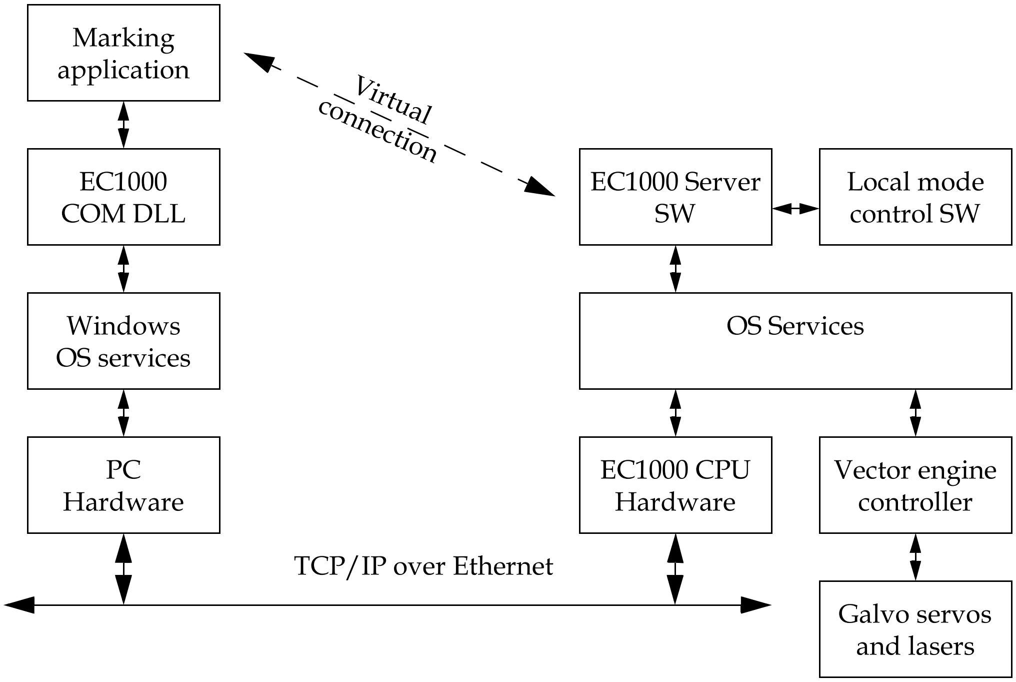


Figure 26 EC1000 System Architecture

5.1 Hardware Overview

The EC1000 is a single board multi-processor system that contains a supervisory/communications control processor, and a high-performance FPGA responsible for real-time micro-vectoring and laser control. The EC1000 is normally paired with an I/O expansion module that takes signals from the EC1000 high-density ribbon cables and re-distributes them to function specific connectors that are easy to interface to. The schematic of the EC1000-IO module is in Appendix A and may be used as a reference for OEM specific designs that may only need a subset of the signals provided. Figure 27 shows how the EC1000-IO module remaps the signals from the EC1000 board.

The EC1000 takes three voltages: ±15 to ±28V for the analog section, and +5V for the digital sections. Voltages required for the various circuits on the module are regulated down from these supplies. Four mass-terminated ribbon cables connect the EC1000 to the EC1000-IO module. Two are 40-pin 2mm pitch carrying all the system control signals, one is a 26-pin 0.050″ pitch carrying the Ethernet and USB signals, and the other is a 20-pin 0.050 pitch cable carrying the RS232 serial communications signals.

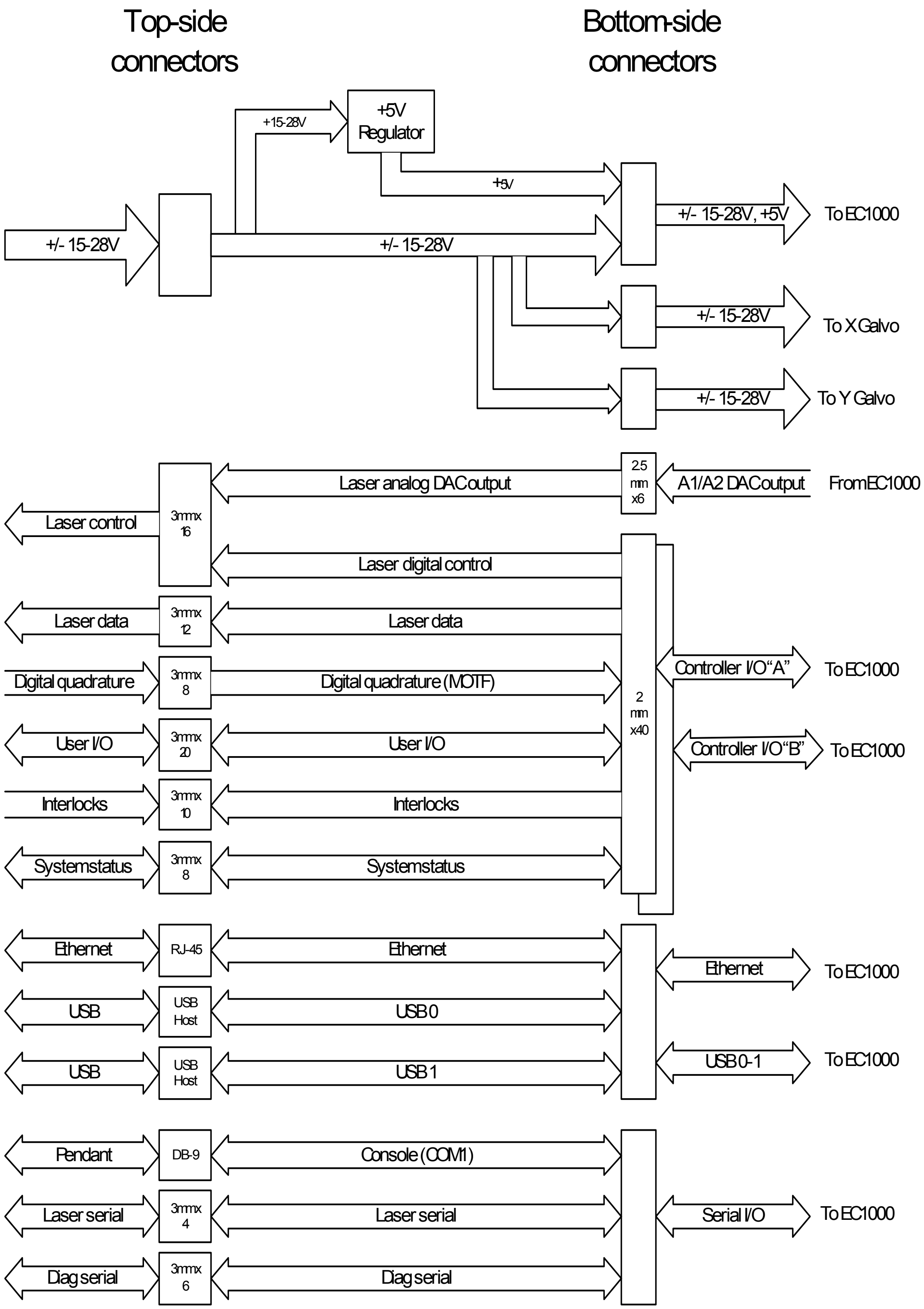


Figure 27   EC1000-IO Module Block Diagram

The EC1000-IO module is intended to be mounted inside a laser head along with the EC1000 main module and presents connectors on the top for interfacing to external devices such as lasers, automation equipment, and factory networks. Connectors on the bottom side are provided for connection to the EC1000 and galvo servo controllers.

When properly configured, the EC1000 boots up from on-board Flash memory, configures any attached devices, and begins local intraction with an attached pendant, or waits for network connections to be made to it.

5.2 Software Overview

The EC1000 contains a fully integrated processor and operating system capable of high-level communications with a supervisory host workstation using TCP/IP protocols, or operating in a fully independent stand-alone mode. The control software of the EC1000 is stored in Flash memory on the module. In a networked application, the EC1000 firmware boots upon system power-up and automatically periodically broadcasts identification information on the network. Application software on a host that links with the EC1000 ActiveX/COM interface software can accept and process these broadcast messages. The broadcast messages contains data that identifies the serial number, friendly name, and IP address of the EC1000. This data, in turn is used to establish session communication channels to the controller. Figure 28 illustrates this relationship.

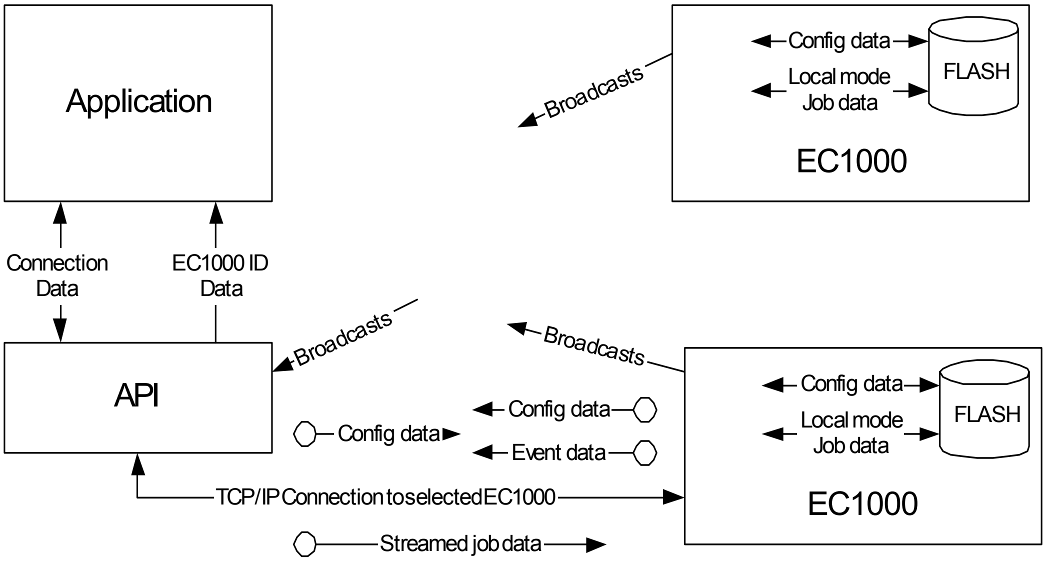


Figure 28 EC1000 Software Data Flow

A communications session, also supported by the COM interface, permits the transmission of job data to the EC1000 and the reception of job-generated messages. Jobs are streamed to the EC1000 with multiple levels of buffering to guarantee full marking performance without CPU load-dependent timing anomalies. Two additional channels of communications are provided to permit asynchronous job aborts, job pausing and resuming, and exception message propagation back to the application.

The COM interface provides high-level functions for setting laser timing and scanner parameters, and for specifying motion vector sequences at any desired speed. Instead of requiring a sequence of mark and jump instructions to be issued by the application, the interface supports the passing of marking objects that are lists of X, Y and Z coordinates. To make such an interface programming language independent, the data of a marking job are transferred across the application/COM interface as XML data. XML is a standard text-based specification language used in many internet applications to represent data in a portable manner.

The system also supports the concept of fixed config data, i.e. data that defines the configuration of the scan-head and surrounding electronics. Examples of such data are lens correction tables, laser interface signal polarities, lens field-size, focal length and calibration values, etc. This data can be set by a system integrator and stored in Flash memory on the EC1000. This data is also specified in XML.

5.3 Scanning Job Fundamentals

The purpose of scanning jobs is to direct the motion of laser galvanometers while simultaneously modulating a laser beam. The laser is turned on when a pattern is to be drawn, and off when moving to the beginning of a new pattern location. In laser marker systems, the drawing action is comonly refered to as a “mark”, and a move to new pattern location is called a “jump”. These terms will be used in the rest of this manual to describe these fudamental actions even though an EC1000 could be used for laser projection where a more appropriate term for “mark” might be “display”.

5.3.1 Coordinate system conventions

Both of the basic movement commands, "mark" and "jump" are expressed in a cartesian coordinate system that is illustrated in Figure 29

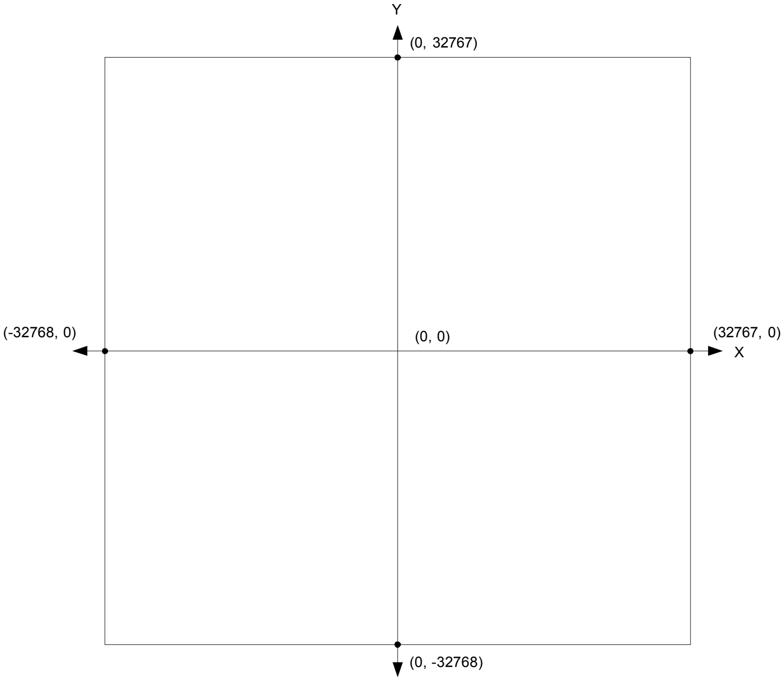


Figure 29 Scanning system coordinate conventions

The imaging field is addressed using 16-bit integers with a range of -32768 to +32767. These units are referred to in the following sections as "bits". All job coordinates are expressed in these units. If an application desires to represent coordinates in other units such as mm, then those coordinates must be scaled appropriately taking into account the projection system optics that are involved.

5.3.2 Marks and Jumps

Laser marking is specified by a list of XML data that defines "jumps" to locations and "marks" to the end points of a vector or series of "connected" vectors otherwise known as poly-vectors. Other XML data represent commands to specify related actions and pauses required to ensure the desired marking quality. The terms Mark, Jump, and related delays are defined below.

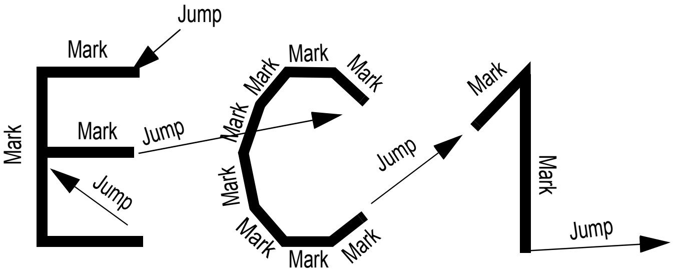


Figure 30 Laser marking sample

Figure 30 shows a sample of the beginning of a simple laser marking. The image is composed of straight line segments (vectors). Connected line segments are formed with sequential *Mark* commands and spaces between unconnected segments are formed with *Jump* commands. Both *Marks* and *Jumps* are controlled-velocity coordinated X & Y galvo motions. The speeds are controllable within a job.

Basic Action Commands

Command/Parameter	Purpose
Jump	A jump causes a (typically) rapid movement of the scanner mirrors to a new position. Ideally no marking occurs during a jump, and typically, the laser is turned off during a jump. The jump command defines the starting point (X and Y coordinates) of the laser marking: the EC1000 directs the laser to the end of the "jump" position where marking will begin.
JumpSpeed	Determines the speed of the jump. The laser is off during a jump and the jump speed is set high enough to maximize throughput, but low enough to minimize instability in the galvo motion as the galvo slows down in its approaches the next marking location.
Mark	A mark command begins the marking process. The laser typically turns on at the beginning of the mark command and continues at a set speed to it's pre-defined location (X and Y coordinates) of the end point of a mark command. As show in Figure 30, subsequent mark commands can create a sequence of marks. The laser is turned off at the end of the last Mark command in a series of commands.
MarkSpeed	Sets the speed during marking. The speed is set to a value such that the laser forms the proper width and depth of a mark in the target media. This is laser power and target material dependent.
Delays	Delays are used to ensure that the marking is complete with no skips, no over-burns, and no inadvertent marks. Delay commands are necessary to fine-tune system control, as need to compensate for system inertia, acceleration, deceleration, and requested jump and marking speeds.

In addition to the dynamic signals used to control the galvanometers and lasers, the EC1000 provides supplemental digital inputs and outputs for external equipment synchronization, and two analog outputs for laser power adjustment. These signals can be manipulated at any point in a job, but are less tightly controlled in time as compared with the galvanometer and laser control signals.

The initial galvanometer position after system power-up is the center of the image field. Marks and jumps are specified from the current position of the galvanometers to a new target position. Jobs typically begin with an absolute jump to the first marking position, and after that, each vector (jump or mark) starts at the new current position, which is usually the end point of the preceding vector.

5.3.3 Micro-vectoring

Controlled velocity marking and jumping is accomplished through a process call micro-vectoring. This process is illustrated in the Figure 31 The marking engine of the EC1000 takes a vector and divides it into multiple shorter segments that are applied to the galvos at regularly spaced time intervals. This interval is known as the update interval. The galvo speed is controlled by magnitude of the *change* in the ouput command at each update period.

The figure shows the sequence of typical output commands for the X axis. The commands for the Y and Z axes are similar and are strictly locked in time with the X axis, differing only in magnitude of the disctrete steps. As the X axis reaches successive targets X_1,X_2, etc., so do the Y and Z axes reach their corresponding targets, Y_1, Z_1,Y_2, Z_2, etc.

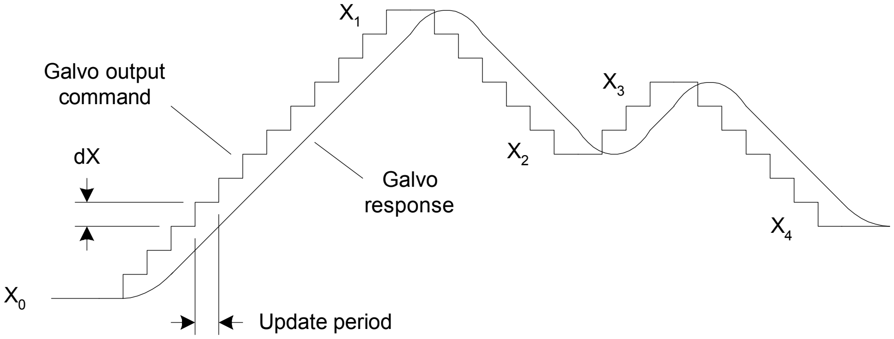
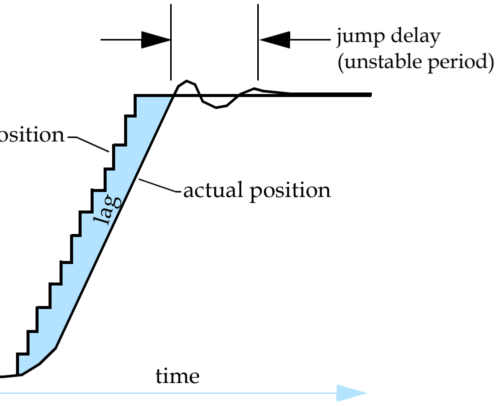
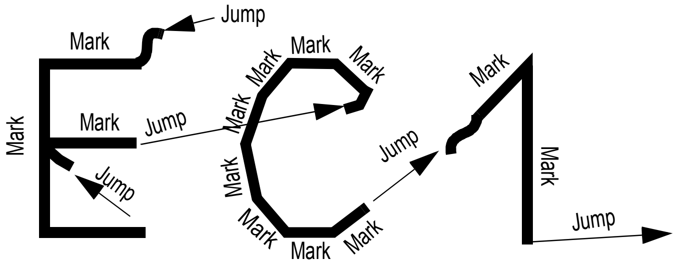
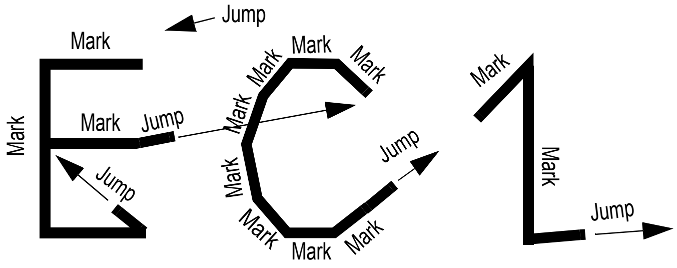


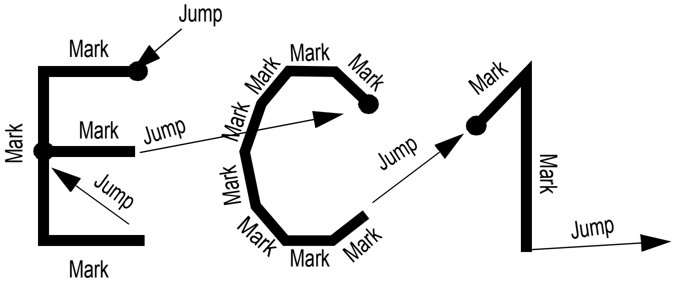
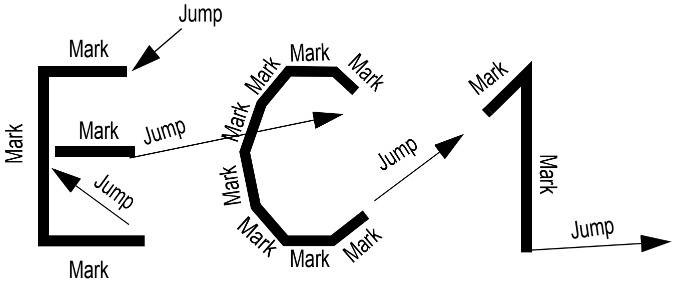
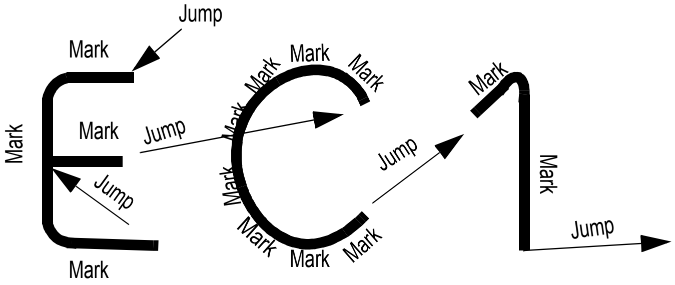
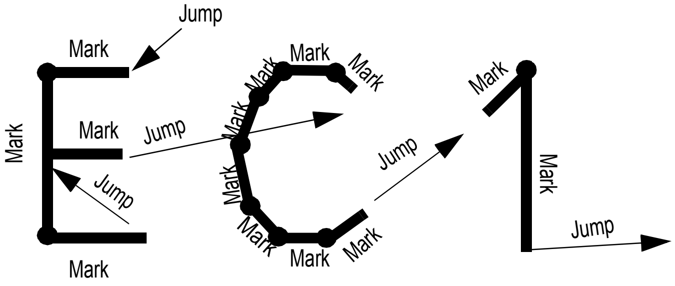
Figure 31 Micro-vector operation

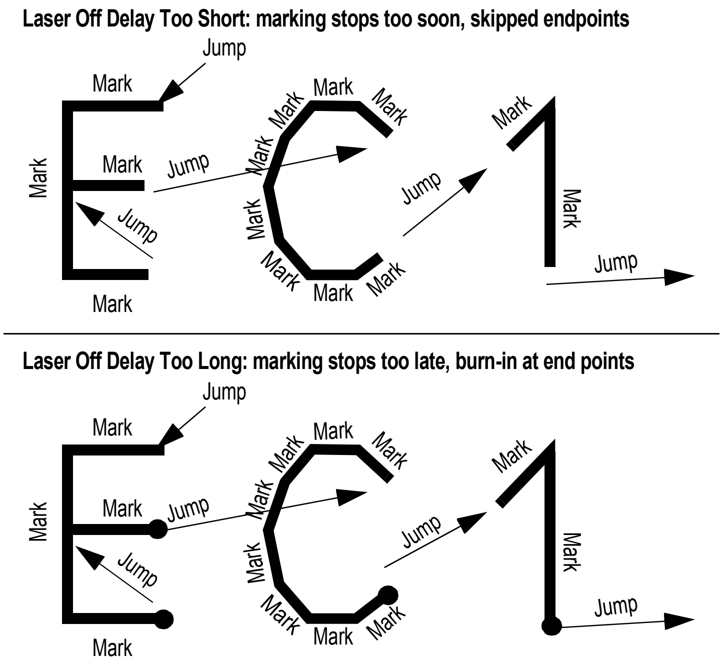
5.3.4 Delays

Because laser scanning systems are electro-mechanical in nature, various delays must be employed to compensate for inertial effects of the mirror and motor structure. These effects generally result in a positional lag of the deflection mirrors relative to the electrical command to make them move. These delays are used to properly time laser on/off and modulation signals relative to the mirror positions. In addition to compensating for lag times, the delays can be used to compensate for transient instability in mirror positions after a step to a new location. The following figures illustrate these effects.

Each system configuration requires fine-tuning of delay commands to ensure full and complete marking with no overburns. The individual delay settings are dependant on the dynamic response of the galvo/mirror combination in use, and the sensitivity characteristics of the marking medium. Determining these delays is typically a trial-and-error process. The delays are specified as part of the job definition described in the next section.

Parameter	Purpose	Effects
JumpDelay	During a jump, the system mirrors accelerate to rapidly get to the next mark position, ideally at the fastest speed possible to minimize overall marking time. As with all accelerations, mirror and system inertia create a slight lag at the beginning of the acceleration. Likewise, the system will require a certain delay (settling time) at the end of the jump as it decelerates to precisely the correct speed required for accurate marking. Acceleration and deceleration times and settling times will vary from system to system (weight of mirrors, type of galvanometer, etc.), and will vary depending on the requested jump speed and the length of the jump. Too short of Jump Delay will cause marking to start before mirrors are properly settled, resulting in inadvertent marking. Too long of a Jump Delay will have no visible effect, but marking is delayed so overall job production time (marking time) increases.	**Jump Command:**  jump delay (unstable period) set position actual position lag time **Jump Delay Too Short: Marking starts before mirrors properly settle**  Jump Mark Mark Mark Mark Mark Jump Mark Mark Jump Jump Mark Mark Mark Mark Mark Jump
MarkDelay	A mark delay at the end of marking a line segment allows the mirrors to move to the required position prior to executing the next mark command. Too short of a Mark Delay will allow the subsequent jump command to begin before the system mirrors get to their final marking position. The end of the current mark will turn upwards towards the direction of the jump vector, as shown to the right. Too long of a Mark delay will cause no visible marking errors, but will add to the overall processing time.	**Mark Delay Too Short: Marking continues into a jump vector**  Jump Mark Mark Mark Mark Mark Jump Mark Mark Jump Jump Mark Mark Mark Mark Mark Jump

LaserOnDelay	The Laser On Delay can be used to prevent burn-in effects at the start of a vector. This delay in time before the laser is turned on is typically used to turn on the laser after the first few microsteps of a mark command to ensure that the laser's motion control systems (mirrors, etc.) are "up to speed" before marking. The vectors must be scanned with a constant velocity to ensure uniform marking. This delay can have either a positive or negative value and will vary with different marking media (some media require a burn-in time to begin marking). The goal is to adjust the LaserOn Delay to ensure uniform marking with no variations of intensity throughout the desired vector. Typically, too short of a delay will cause burn-in effects, and too long of a delay will cause skipping (missed line segments).	**Laser On Delay Too Short: burn-in at start points**  Jump Mark Mark Mark Jump Jump Mark Mark Mark Mark Mark Mark Mark Mark Jump Mark Mark Jump **Laser On Delay Too Long: marking starts too late, skips points**  Jump Mark Mark Mark Jump Jump Mark Mark Mark Mark Mark Mark Mark Mark Jump Mark Mark Jump
PolyDelay	A polygon delay is a delay automatically inserted between two marking segments. The minimum delay allows enough time for the galvos and mirror to "catch-up" with the command signal before a new command is issued to move on to the next point. If variable polygon delay mode is selected, then the delay is variable and changes as a function how large an angular change is required to move on to the next point. The larger the angular change, the longer it takes for the galvos to change direction and accelerate to the required speed in the new direction. The delay is scaled proportionally to the size of the angle.	**Polygon Delay Too Short: characters not wellformed**  Jump Mark Mark Mark Jump Jump Mark Mark Mark Mark Mark Mark Mark Mark Jump Mark Mark Jump **Polygon Delay Too Long: burn-in at junctions in the vector**  Jump Mark Mark Mark Jump Jump Mark Mark Mark Mark Mark Mark Mark Mark Jump Mark Mark Jump

<table>
<tr>
<td>LaserOffDelay</td>
<td>
<p>The Laser Off Delay can be used to prevent burn-in effects at the end of a vector. This delay in time before the laser is turned off is typically used to turn off the laser just before the last few microsteps of a mark command to ensure that the marking stops exactly where it is desired to stop.</p>
<p>The goal is to adjust the Laser Off Delay to ensure uniform marking with no variations of intensity throughout the desired vector.</p>
<p>Typically, too short of a delay will cause skipping of line segments, and too long of a delay will cause burn-in at the end of line segments.</p>
</td>
<td>


</td>
</tr>
</table>

The relationship of the delays to the micro-vectoring process is illustrated in Figure 32.

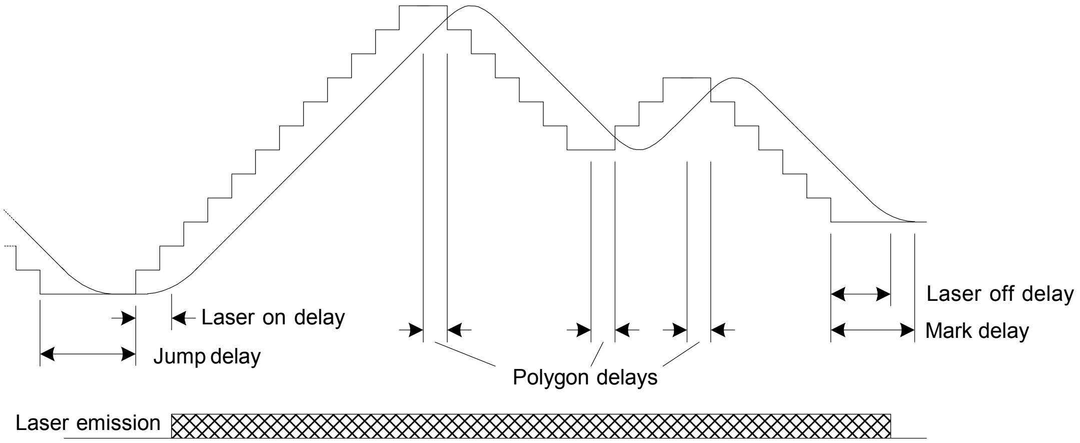


Figure 32 Micro-vectoring and laser timing relationships

5.4 Image Field Correction

Image field correction capability is provided to compensate for optical errors induced by all two-mirror laser beam systems. These optical distortions are caused by a number of factors, including the distance between each mirror, the distance between the mirrors and the image field, and the type of lens used in the laser for focusing the laser beam.

Figure 33 shows the basic projection system layout.

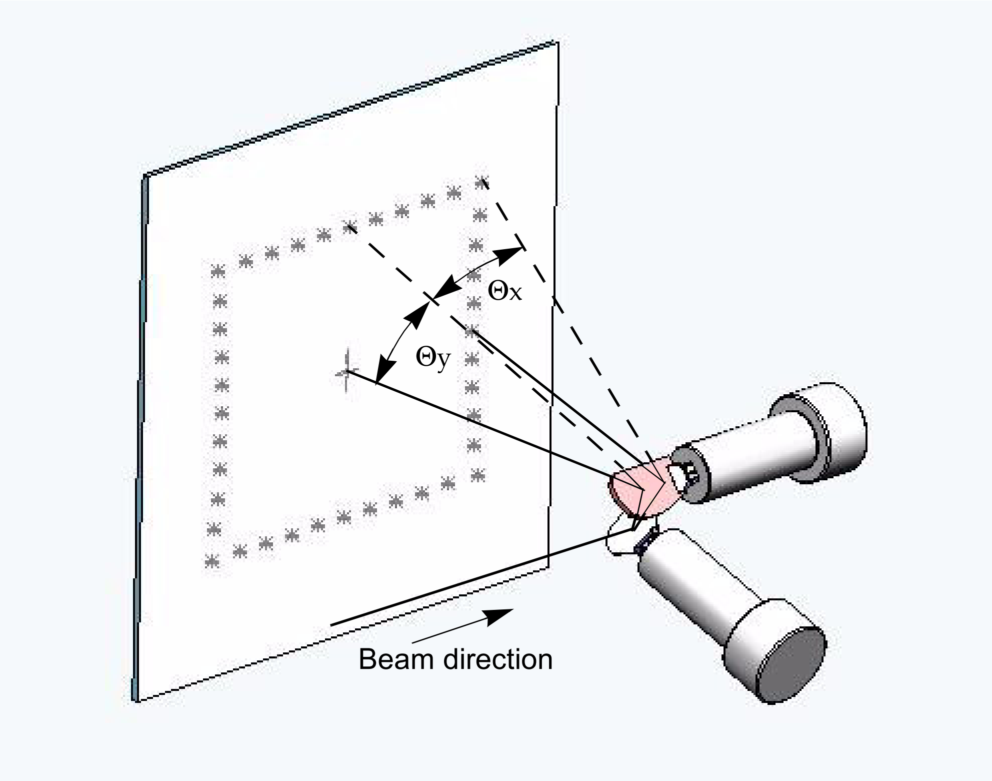


Figure 33 Projection system layout

5.4.1 X-Y Mirror Induced Distortion

Projection of a laser beam via an X-Y mirror set controlled by galvanometers induces distortion in the X axis propotional to the tangent of the angle of the Y axis mirror and the distance from the focal plane to the center of the Y axis mirror. This distortion is also known as “pincushion” distortion.

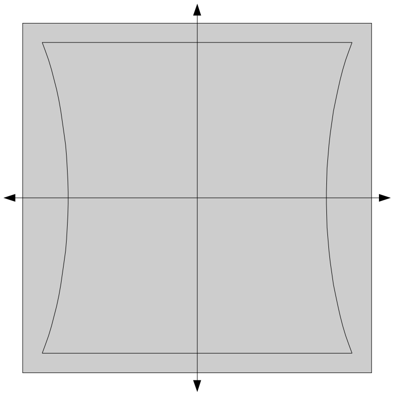


Figure 34 Pincushion distortion caused by X-Y mirror set

5.4.2 F-theta Objective Induced Distortion

The addition of an F-theta objective in the laser field provides direct proportionality between the scan angle and the distance in the image field, as well as ensure that the focus lies on a flat surface. F-theta objective lenses, like all optical lenses, are not perfect and induce their own projection field distortions. This distortion, illustrated in Figure 35, is called "pillow" distortion for what it does to a square image. In reality, this distortion is radially symetric from the image field origin and can often be modeled as a third order polynomial. Many projection lens vendors will provide these model coefficients, or measurement data from which these coefficients can be derived. For many applications, however, this distortion is negligible.

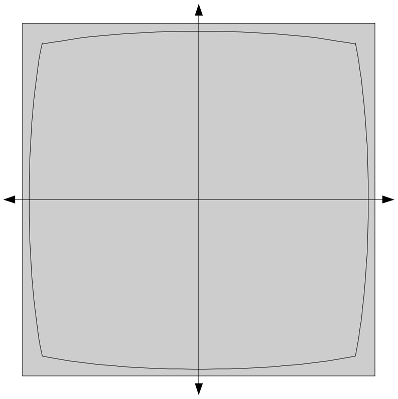


Figure 35 Pillow distortion caused by F-theta lens

5.4.3 Composite Distortion and Correction Methodology

The two distortion components described above combine to to create a distorted image field similar to that shown in Figure 36. This distortion is automatically compensated for by the EC1000 through the use of correction tables.

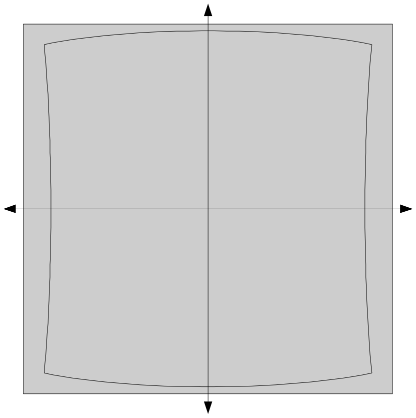


Figure 36 Composite Image Field Distortion

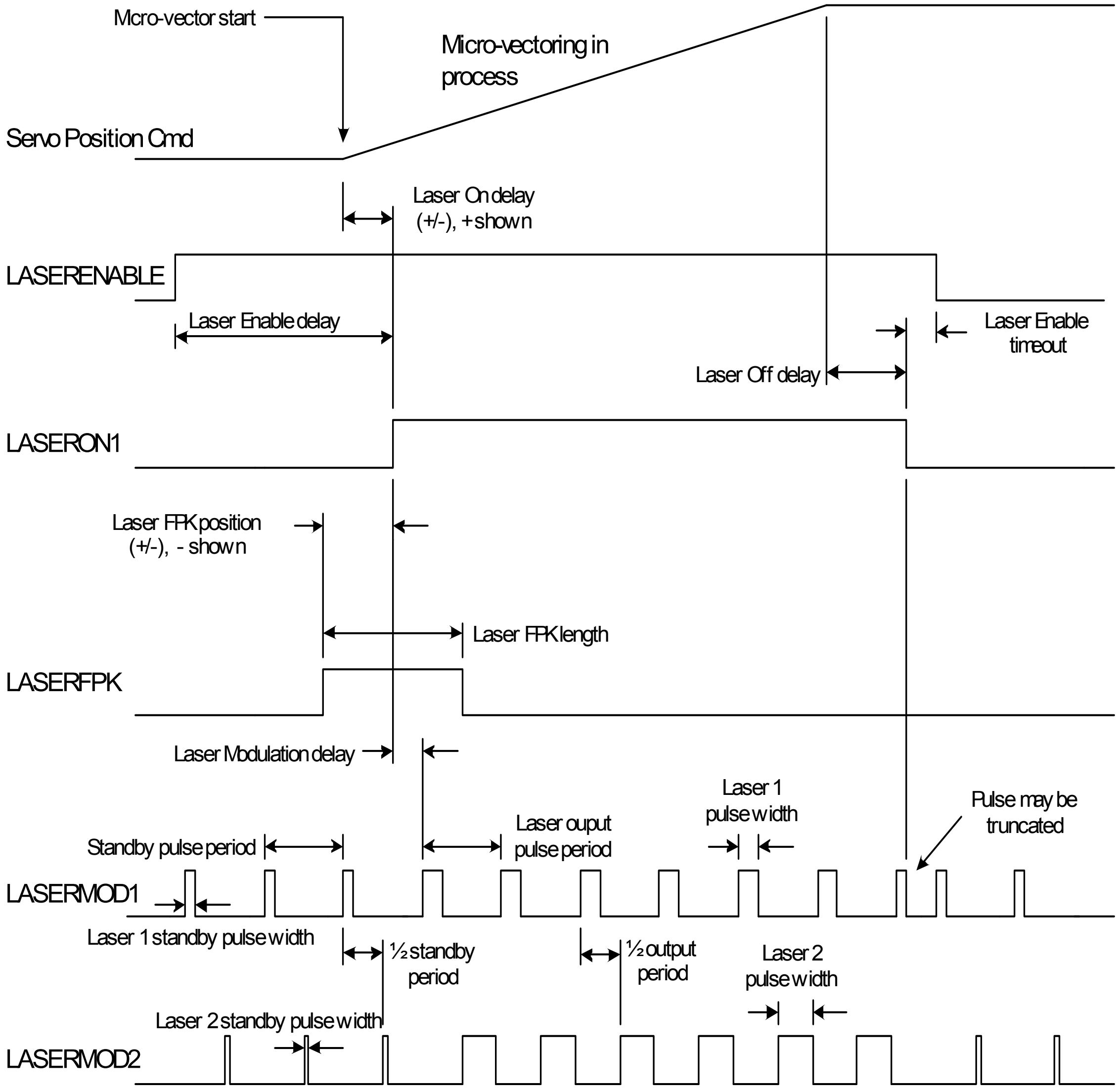
Correction tables represent a 65x65 element grid covering the full addressable projection range of the system. Each grid element contains three correction components: one each for the X, Y and Z axes. The components represent an offset that if added to an ideal position command for that point, would alter the galvo positions such that the resulting projected point would fall onto a "perfect" grid, i.e. the point would be "corrected".

During the micro-vectoring process at each update interval, the EC1000 calculates the ideal position of the mirrors along the path. It compares this value to the correction table grid and accesses the four grid points that immediately surround the calculated point. The corrections at these four points are proportionally averaged depending on how close the ideal point is to each grid point. This process, called bi-linear interpolation, produces a correction that is applied to the ideal point, and the result is then sent to the system D/A converters and serial digital command outputs.

The EC1000 has integral support for up to four independent three-axis correction tables. These tables are organized in pairs where the first table of the pair is applied to the analog D/A converters, and the second is applied to the XY2-100 port. The table pairs are selectable dynamically though the job parameter *ActiveCorrectionTable* described in section 6.3.5. Table contents can be automatically loaded on board power-up from stored correction table files, or can be dynamically loaded via the *sendStreamData* method of the session API described in section 6.3.

5.5 Laser Timing Control

The EC1000 provides very flexible laser control capability that is synchronized with galvo motion control. Six[1] dedicated TTL compatible signals are provided at all times whose timing relationships are defined by the diagram below. Not all signals may be required for a given customer laser configuration. An integrator need only select an appropriate subset of these signals, and configure them via software with appropriate timing parameters. Provisions are made for the synchronous control of two separate lasers running with two independent pulse-widths during the laser-on period. Laser control timing is specified in terms of laser timing "ticks" which can be set via software to an interval as small as 20ns to as large as 1.3ms with a resolution of 20ns. The typical tick value is set to 1us.



Notes:

1. Laser Enable delay, Laser Enable timeout, and Laser Modulation delay must be >= 0
2. Laser Enable delay is relative to the leading edge of LASERON but the leading edge of LASERENABLE will never occur after:
   a) Mcro-vector start
   b) the leading edge of LASERON
   c) the leading edge of LASERFPK
3. Laser On delay may be positive or negative and is relative to Mcro-vector start
4. Laser FPK position may be positive or negative and is relative to the leading edge of LASERON
5. Laser pulse generation starts relative to but no earlier than the leading edge of LASERON or the leading edge of LASERFPK
6. Standby pulse suppression is accomplished by setting the standby pulse width to zero
7. The first laser-on laser pulse on LASERMOD1 & 2 is always a full pulse

Figure 37 Laser timing relationships

[1] The signal LASERON2 is also provided with multiple progammable functions to support pointer laser operation

Figure 37 introduces 12 timing parameters that can be set to yield signal relationships that are suitable for controlling all known commercial lasers used in marking or projection scanning systems. The reference point for the timing is the beginning of micro-vectoring shown on the diagram as Micro-vector start. When the marking engine processor encounters a mark instruction it asserts the LASERENABLE signal and waits the specified Laser Enable delay. The LASERENABLE signal is normally used to precondition fiber laser systems in anticipation of being called into action during a marking operation. LASERENABLE will remain asserted until the Laser Enable timeout period expires after marking has stopped, i.e. after the last vector of a sequence of marking vectors. If a new series of marking vectors begins before the Laser Enable timeout expires, LASERENABLE remains asserted and a new timeout period is armed.

When the Laser Enable delay expires, one of three things will happen based on the setting of the delay parameters:

1. Micro-vectoring begins if Laser On delay and Laser First Pulse Killer (FPK) position are both positive
2. LASERON is aserted if Laser On delay is negative and Laser FPK position is positive
3. LASERFPK is aserted if Laser FPK delay is negative and Laser On delay is also negative OR if Laser FPK delay is negative and the absolute value of Laser FPK delay is larger than Laser On delay if Laser On delay is positive

As can be seen from the diagram the timing of laser emission is directly related to the timing of the LASERON signal. Pulse emission will never occur earlier than the leading edge of LASERON or LASERFPK, but may be delayed after the leading edge of LASERON by setting the Laser Modulation delay to a non-zero value. The LASERFPK signal may be asserted any time before or after the leading edge of LASERON. The signals LASERFPK and LASERMODn are dependently related to the timing of LASERON. That is, if Laser On delay is changed, the system timing is changed to keep all three signals in the proper timing relationship.

The LASERMOD1 and LASERMOD2 signals are time-related in that the periods of the signals must be the same for the standby (laser not active) and ouput active (laser emitting) intervals. The phase of the two signal is locked 180 degrees apart from each other to ensure that the two lasers never fire at the same instant of time, thus reducing peak power demands and reducing EMI effects. Otherwise, the pulse widths during the standby and output active intervals are independent and programmable for each signal.

The lasers are turned off automatically after the micro-vectoring completes and the Laser Off delay expires. The LASERON signal is de-asserted and the LASERMOD1/2 signals switch to the standby mode.

5.5.1 Software Control of Laser Timing

The laser timing configuration is statically specified in an XML based configuration file stored on the EC1000 and is automatically applied at system boot-up. The configuration can be changed by reading it through the software Application Programming Interface (API), altering it, and then sending it back to the controller. Changes made this way would be applied every time the EC1000 re-initializes. The configuration information can also be specifed dynamically in a job stream and applied on a temporary basis being persistant only until the next re-initialization. These concepts are described more fully in section 6 Application Programming Interface.

All of the programmable control elements of the EC1000 are manipulated through XML language constructs passed through the API. At system boot-up, XML configuration files are read from Flash memory on the controller and some of the parameters are applied to the hardware to pre-configure it. The Laser Configuration fixed-data described in section: Laser Configuration, contains definitions to specify laser marking and idle-time pulse-widths and frequency, signal polarities, FPK signal timing, etc. These parameters do not often change during a marking job, although provisions are made in the Job Stream XML specification to do so if required. Other laser timing parameters such an Laser On Delay and Laser Off Delay are expected to change as the job is tuned for best performance. These parameters are directly controlled by JobStream XML constructs, but not in the Laser Configuration XML specification.

Table 2: Laser Configuration Control XML with example settings

Static Configuration XML	Dynamic Configuration XML	Example Description
<LsrTiming>**50**</LsrTiming>	<set id=”LaserTiming”>**50**</set>	Set the laser time base to 1µsec: 50 * 20ns = 1µsec “tick”
<LsrPipeDly>**0**</LsrPipeDly>	<set id=”LaserPipelineDelay”>**0**</set>	Normally zero except when using CTI DC900 or DC2000 digital servos. This value is used the delay all of the the laser timing signals as a group relative to the galvo commands.
<LsrPwrDly>**1700**</LsrPwrDly>	<set id=”LaserPowerDelay”>**1700**</set>	The job will delay for 1.7msec every time the laser power is changed
<LENAHigh>**false**</LENAHigh> <LONHigh>**false**</LONHigh> <LON2High>**false**</LONHigh> <LMOD1High>**false**</LMOD1High> <LMOD2High>**false**</LMOD2High> <LFPKHigh>**false**</LFPKHigh>	<set id=”LaserModeConfig”>**0**</set>	LaserModeConfig uses a bit-mask to represent the various signal polartities. See section: Laser Configuration for more details.
<LsrEnaDly>**7000**</LsrEnaDly>	<set id=”LaserEnableDelay”>**7000**</set>	Wait 7msec after asserting the LASERENABLE signal
<LsrEnaTmo>**4000**</LsrEnaTmo>	<set id=”LaserEnableTimeout”>**4000**</set>	Deassert LASSERENABLE if there is no laser activity requested within 4msec of when the laser turned off.
<LsrModDly>**20**</LsrModDly>	<set id=”LaserModDelay”>**20**</set>	Delay the modulation of the laser for 20 laser timing ticks (20µsec) after LASERON is asserted
<FpsPos>**-30**</FpsPos> <FpsWidth>**10**</FpsWidth>	<set id=”LaserFPK”>**--30,10**</set>	Assert LASERFPK -30 laser timing ticks (-30µsec) relative to the leasding edge of LASERON. Deassert LASERFPK 10 laser timing ticks (10µsec) after it was asserted.
<TickleWidth1>**5**</TickleWidth1> <TickleFreq1>**5**</TickleFreq1>	<set id=”LaserStandby”>**1, 5, 200**</set>	For Laser 1, set the stand-by (idle) pulse width to 5 laser timing ticks (5µsec) and set the period to 200 ticks (200µsec). This is a pulse frequency of 5KHz
<TickleWidth2>**10**</TickleWidth2> <TickleFreq2>**5**</TickleFreq2>	<set id=”LaserStandby”>**2, 10, 200**</set>	For Laser 2, set the stand-by or idle pulse width to 10 laser timing ticks (10µsec) and set the period to 200 ticks (200µsec). This is a pulse frequency of 5KHz. Pulse period/freq must be the same as for Laser 1
N/A	<set id=”LaserOnDelay”>**150**</set>	LASERON is asserted 150 laser timing ticks (150µsec) after the start of micro-vectoring
N/A	<set id=”LaserOffDelay”>**100**</set>	LASERON is deasserted 100 laser timing ticks (100µsec) after the micro-vectoring has completed
N/A	<set id=”LaserPulse”>**1, 8, 15**</set>	For Laser 1, set the “Laser On” pulse width to 8 laser timing ticks (8µsec) and set the period to 15 ticks (15µsec). This is a pulse frequency of 66.7KHz
N/A	<set id=”LaserPulse”>**2, 10, 15**</set>	For Laser 1, set the “Laser On” pulse width to 10 laser timing ticks (10µsec) and set the period to 15 ticks (15µsec). This is a pulse frequency of 66.7KHz. Pulse period/freq must be the same as for Laser 1

5.5.2 Laser Timing Emulation

Traditional laser scanning controllers often use fixed signal sets and constrained timing relationships to provide laser control, whereas the EC1000 uses a completely flexible and programmable suite of signals. The EC1000 can be configured to emulate the timing produced be other commercial controllers because of the flexible nature of the laser timing generator.

Typical laser configurations are shown in the following diagrams. These configurations emulate the laser control performed by the RAYLASE AG SP-ICE card, and SCANLAB RTC3/4 and SCANalone series of scan head controllers. These configurations are by no means the only ones possible and new laser systems are frequently introduced. Most notably, fiber lasers have become much more reliable and affordable offering compact packaging and highly efficient energy properties. The EC1000 has been specifically designed to accomodate the unique timing requirements of these lasers.

Along with each diagram, examples of the XML for both statically and dynamically configuring the behavior is illustrated. Only those parameters that are meaningful for the illustration are specified in the examples. Other parameters used to set signal polarities, Laser Enable Delay/Timeout, Standby (Tickle) timing, Laser Power Delay and Laser Pipeline Delay are almost always set to pre-defined values. Laser Pulse timing, although potentially variable during a job, does not affect the fundamental signal relationships that define the laser emulation modes. In addition, the specification of a laser timing "tick" is most conveniently set to a 1µsec interval, which is assumed in the examples.

CO_2 Laser Timing

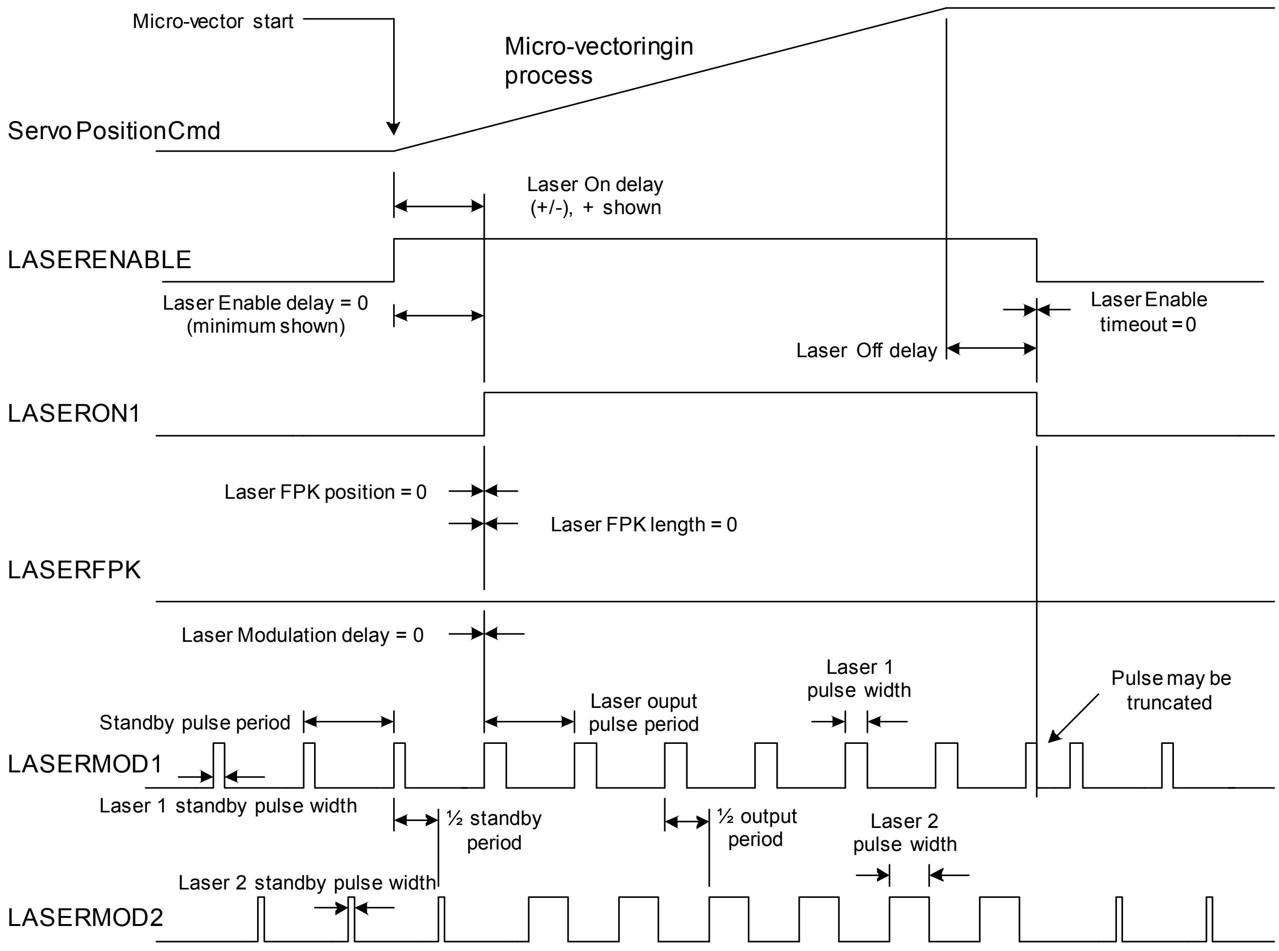


Figure 38 Laser timing for CO_2 laser systems

The simplest emulation mode is for CO_2 lasers. These lasers do not require a Laser FPK signal so these parameters are set to zero. LASERENABLE is also not typically needed therefore the Laser Enable delay and Laser Enable timeout can be set to zero to maximize throughput. In fact, whenever LASERENABLE is not required, the Laser Enable delay should be set to zero.

Table 3: Example CO_2 Laser Configuration XML

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<LsrEnaDly>**0**</LsrEnaDly>	<set id="LaserEnableDelay">**0**</set>	Maximizes throughput
<LsrEnaTmo>**0**</LsrEnaTmo>	<set id="LaserEnableTimeout">**0**</set>	Maximizes throughput
<LsrModDly>**0**</LsrModDly>	<set id="LaserModDelay">**0**</set>	No modulation delay required
<FpsPos>**0**</FpsPos>	<set id="LaserFPK">**0, 0**</set>	No FPK required
<FpsWidth>**0**</FpsWidth>		
<TickleWidth1>**5**</TickleWidth1>	<set id="LaserStandby">**1, 5, 200**</set>	Laser 1 stand-by; pulse width == 5 laser timing ticks (5µsec); pulse period == 200 ticks (200µsec) == 5KHz
<TickleFreq1>**5**</TickleFreq1>		
<TickleWidth2>**10**</TickleWidth2>	<set id="LaserStandby">**2, 10, 200**</set>	Laser 2; pulse width = 10 laser timing ticks (10µsec); pulse period == 200 ticks (200µsec) == 5KHz, must be same as Laser 1
<TickleFreq2>**5**</TickleFreq2>		
N/A	<set id="LaserOnDelay">**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id="LaserOffDelay">**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id="LaserPulse">**1, 8, 15**</set>	Laser 1 operating; pulse width == 8 laser timing ticks (8µsec); pulse period == 15 ticks (15µsec) == 66.7KHz
N/A	<set id="LaserPulse">**2, 10, 15**</set>	Laser 2 operating; pulse width == 10 laser timing ticks (10µsec); pulse period == 15 ticks (15µsec) == 66.7KHz, must be same as Laser 1

Nd:YAG Emulation Mode-1 Timing

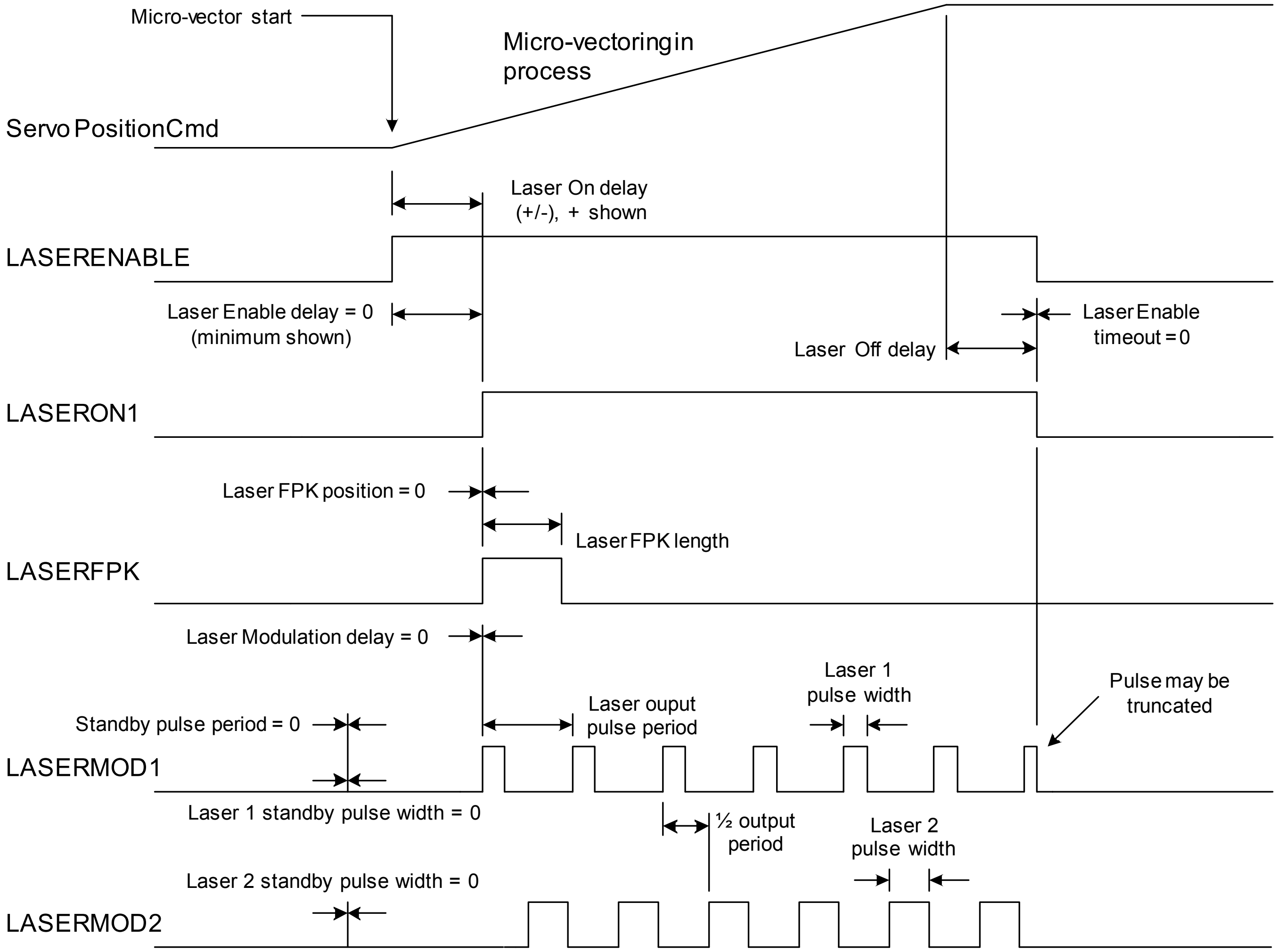


Figure 39 Nd:YAG Emulation Mode-1 (Raylase Nd:YAG Mode-1 and Scanlab YAG 1)

Most of theYAG modes do not require standby or idle pulses. To supress these pulses, the Standby pulse width and pulse period are set to zero. In this mode, the LASERFPK is asserted coincident with the LASERON and LASERMOD signals, but its assertion can have variable length. If the Laser On delay is modified, the timing of LASERFPK and LASERMOD track with it.

Table 4: Example Nd:YAG Mode-1 Laser Configuration XML

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<LsrEnaDly>**0**</LsrEnaDly>	<set id=”LaserEnableDelay”>**0**</set>	Maximizes throughput
<LsrEnaTmo>**0**</LsrEnaTmo>	<set id=”LaserEnableTimeout”>**0**</set>	Maximizes throughput
<LsrModDly>**0**</LsrModDly>	<set id=”LaserModDelay”>**0**</set>	No modulation delay required
<FpsPos>**0**</FpsPos>	<set id=”LaserFPK”>**0, 15**</set>	Example FPK length set to 15usec with no shift
<FpsWidth>**15**</FpsWidth>		
<TickleWidth1>**0**</TickleWidth1>	<set id=”LaserStandby”>**1, 0, 0**</set>	1 == laser; No tickle pulses required
<TickleFreq1>**0**</TickleFreq1>		
<TickleWidth2>**0**</TickleWidth2>	<set id=”LaserStandby”>**2, 0, 0**</set>	2 == laser; No tickle pulses required
<TickleFreq2>**0**</TickleFreq2>		
N/A	<set id=”LaserOnDelay”>**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id=”LaserOffDelay”>**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id=”LaserPulse”>**1, 8, 15**</set>	Laser 1 operating; pulse width == 8 laser timing ticks (8µsec); pulse period == 15 ticks (15µsec) == 66.7KHz
N/A	<set id=”LaserPulse”>**2, 10, 15**</set>	Laser 2 operating; pulse width == 10 laser timing ticks (10µsec); pulse period == 15 ticks (15µsec) == 66.7KHz, must be same as Laser 1

Nd:YAG Emulation Mode-2 Timing

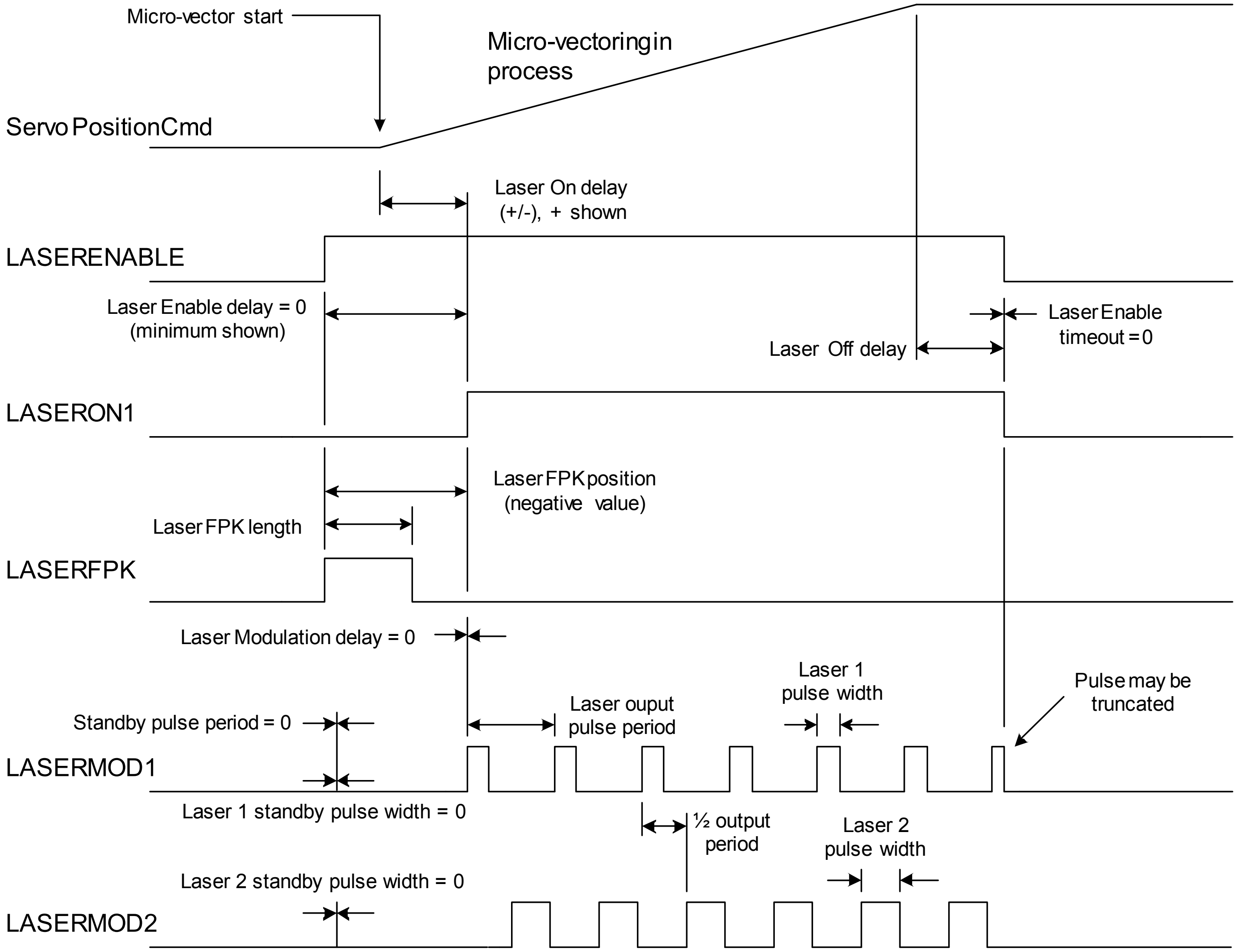


Figure 40 Nd:YAG Emulation Mode-2 (Raylase Nd:YAG Mode-2)

In this mode, the LASERFPK signal is a 10µ sec pulse asserted a variable amount of time prior to the assertion of LASERON and the coincident generation of pulses. This timing is typically suited for Lee and Coherent lasers.

Table 5: Example Nd:YAG Mode-2 Laser Configuration XML

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<LsrEnaDly>**0**</LsrEnaDly>	<set id="LaserEnableDelay">**0**</set>	Maximizes throughput
<LsrEnaTmo>**0**</LsrEnaTmo>	<set id="LaserEnableTimeout">**0**</set>	Maximizes throughput
<LsrModDly>**0**</LsrModDly>	<set id="LaserModDelay">**0**</set>	No modulation delay required
<FpsPos>-20</FpsPos>	<set id="LaserFPK">**-20, 10**</set>	Example FPK length set to 10µsec with a minus 20µsec shift relative to LASERON
<FpsWidth>**10**</FpsWidth>		
<TickleWidth1>**0**</TickleWidth1>	<set id="LaserStandby">**1, 0, 0**</set>	1 == laser; No tickle pulses required
<TickleFreq1>**0**</TickleFreq1>		
<TickleWidth2>**0**</TickleWidth2>	<set id="LaserStandby">**2, 0, 0**</set>	2 == laser; No tickle pulses required
<TickleFreq2>**0**</TickleFreq2>		
N/A	<set id="LaserOnDelay">**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id="LaserOffDelay">**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id="LaserPulse">**1, 8, 15**</set>	Laser 1 operating; pulse width == 8 laser timing ticks (8µsec); pulse period == 15 ticks (15µsec) == 66.7KHz
N/A	<set id="LaserPulse">**2, 10, 15**</set>	Laser 2 operating; pulse width == 10 laser timing ticks (10µsec); pulse period == 15 ticks (15µsec) == 66.7KHz, must be same as Laser 1

Nd:YAG Emulation Mode-3 Timing

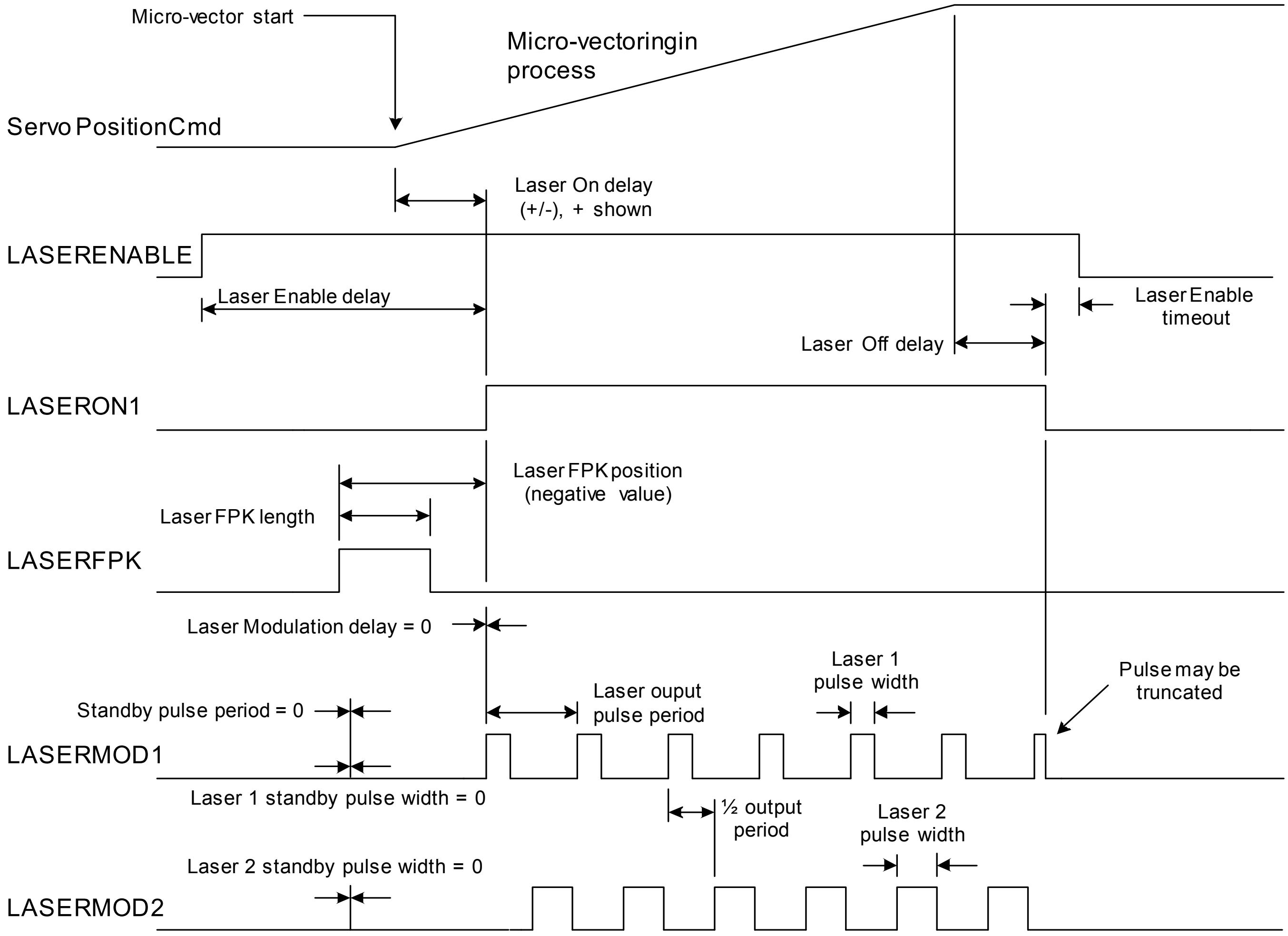


Figure 41 Nd:YAG Emulation Mode-3 (Raylase Nd:YAG Mode-3)

This mode is very similar to Mode-2. The difference is that Laser FPK length can vary. Spectron lasers normally use this type of timing.

Table 6: Example Nd:YAG Mode-3 Laser Configuration XML

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<LsrEnaDly>**0**</LsrEnaDly>	<set id="LaserEnableDelay">**0**</set>	Maximizes throughput
<LsrEnaTmo>**0**</LsrEnaTmo>	<set id="LaserEnableTimeout">**0**</set>	Maximizes throughput
<LsrModDly>**0**</LsrModDly>	<set id="LaserModDelay">**0**</set>	No modulation delay required
<FpsPos>-18</FpsPos>	<set id="LaserFPK">**-20, 18**</set>	Example FPK length set to 18µsec with a minus 20µsec shift relative to LASERON
<FpsWidth>**10**</FpsWidth>		
<TickleWidth1>**0**</TickleWidth1>	<set id="LaserStandby">**1, 0, 0**</set>	1 == laser; No tickle pulses required
<TickleFreq1>**0**</TickleFreq1>		
<TickleWidth2>**0**</TickleWidth2>	<set id="LaserStandby">**2, 0, 0**</set>	2 == laser; No tickle pulses required
<TickleFreq2>**0**</TickleFreq2>		
N/A	<set id="LaserOnDelay">**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id="LaserOffDelay">**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id="LaserPulse">**1, 8, 15**</set>	Laser 1 operating; pulse width == 8 laser timing ticks (8µsec); pulse period == 15 ticks (15µsec) == 66.7KHz
N/A	<set id="LaserPulse">**2, 10, 15**</set>	Laser 2 operating; pulse width == 10 laser timing ticks (10µsec); pulse period == 15 ticks (15µsec) == 66.7KHz, must be same as Laser 1

Nd:YAG Emulation Mode-4 Timing

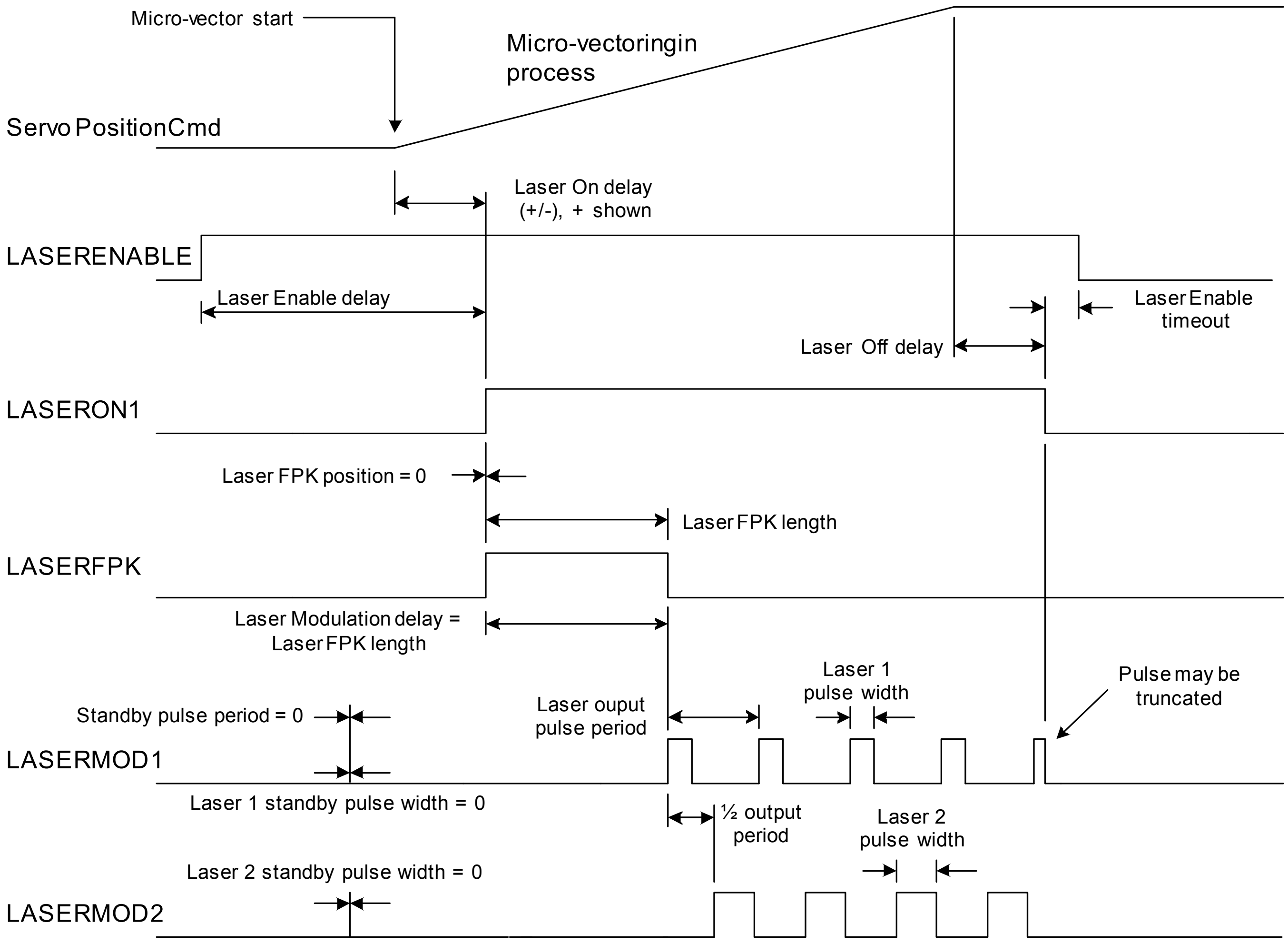


Figure 42 Nd:YAG Emulation Mode-4 (Scanlab YAG 2)

In this mode, the LASERFK signal leading edge is coincident with the leading edge of LASERON and the generation of the laser pulses is delayed to be coincident with the trailing edge of the LASERFPK signal.

Table 7: Example Nd:YAG Mode-4 Laser Configuration XML

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<LsrEnaDly>**0**</LsrEnaDly>	<set id="LaserEnableDelay">**0**</set>	Maximizes throughput
<LsrEnaTmo>**0**</LsrEnaTmo>	<set id="LaserEnableTimeout">**0**</set>	Maximizes throughput
<LsrModDly>**15**</LsrModDly>	<set id="LaserModDelay">**15**</set>	Laser modulation delayed by the same value as the LASERFPK length
<FpsPos>0</FpsPos>	<set id="LaserFPK">**0, 15**</set>	Example FPK length set to 15µsec with no shift relative to LASERON
<FpsWidth>**15**</FpsWidth>		
<TickleWidth1>**0**</TickleWidth1>	<set id="LaserStandby">**1, 0, 0**</set>	1 == laser; No tickle pulses required
<TickleFreq1>**0**</TickleFreq1>		
<TickleWidth2>**0**</TickleWidth2>	<set id="LaserStandby">**2, 0, 0**</set>	2 == laser; No tickle pulses required
<TickleFreq2>**0**</TickleFreq2>		
N/A	<set id="LaserOnDelay">**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id="LaserOffDelay">**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id="LaserPulse">**1, 8, 15**</set>	Laser 1 operating; pulse width == 8 laser timing ticks (8µsec); pulse period == 15 ticks (15µsec) == 66.7KHz
N/A	<set id="LaserPulse">**2, 10, 15**</set>	Laser 2 operating; pulse width == 10 laser timing ticks (10µsec); pulse period == 15 ticks (15µsec) == 66.7KHz, must be same as Laser 1

Nd:YAG Emulation Mode-5 Timing

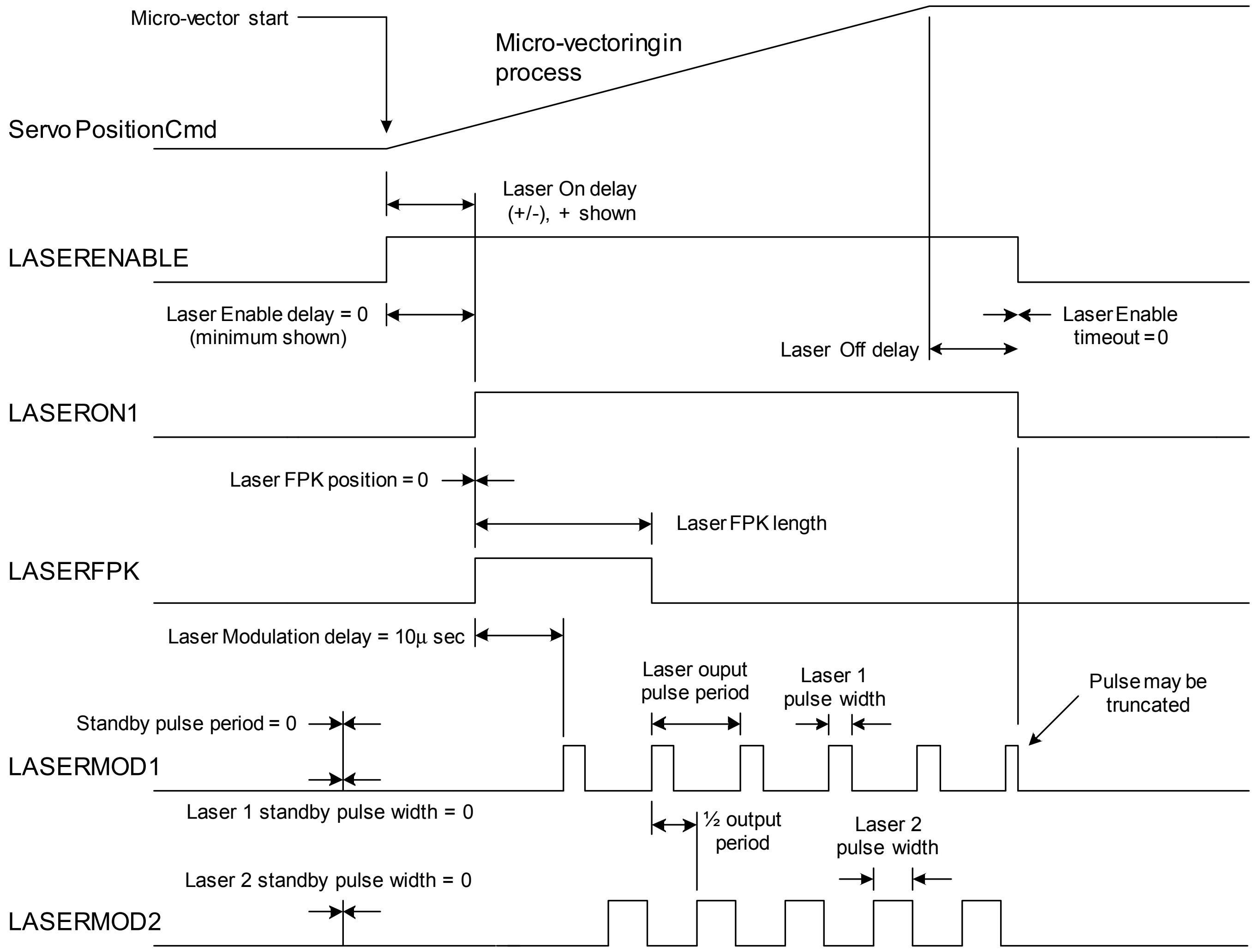


Figure 43 Nd:YAG Emulation Mode-5 (Scanlab YAG 3)

This mode is very similar to emulation mode-4. The difference is that the start of laser pulse generation is 10µ sec after the coincident leading edges of LASERON and LASERFPK.

Table 8: Example Nd:YAG Mode-5 Laser Configuration XML

Static Configuration XML	Dynamic Configuration XML	Example Description
<LsrEnaDly>**0**</LsrEnaDly>	<set id="LaserEnableDelay">**0**</set>	Maximizes throughput
<LsrEnaTmo>**0**</LsrEnaTmo>	<set id="LaserEnableTimeout">**0**</set>	Maximizes throughput
<LsrModDly>**10**</LsrModDly>	<set id="LaserModDelay">**10**</set>	Laser modulation delayed by 15µsec relative to LASERON
<FpsPos>0</FpsPos> <FpsWidth>**20**</FpsWidth>	<set id="LaserFPK">**0, 20**</set>	Example FPK length set to 20µsec with no shift relative to LASERON
<TickleWidth1>**0**</TickleWidth1> <TickleFreq1>**0**</TickleFreq1>	<set id="LaserStandby">**1, 0, 0**</set>	1 == laser; No tickle pulses required
<TickleWidth2>**0**</TickleWidth2> <TickleFreq2>**0**</TickleFreq2>	<set id="LaserStandby">**2, 0, 0**</set>	2 == laser; No tickle pulses required
N/A	<set id="LaserOnDelay">**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id="LaserOffDelay">**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id="LaserPulse">**1, 8, 15**</set>	Laser 1 operating; pulse width == 8 laser timing ticks (8µsec); pulse period == 15 ticks (15µsec) == 66.7KHz
N/A	<set id="LaserPulse">**2, 10, 15**</set>	Laser 2 operating; pulse width == 10 laser timing ticks (10µsec); pulse period == 15 ticks (15µsec) == 66.7KHz, must be same as Laser 1

Fiber Laser Timing

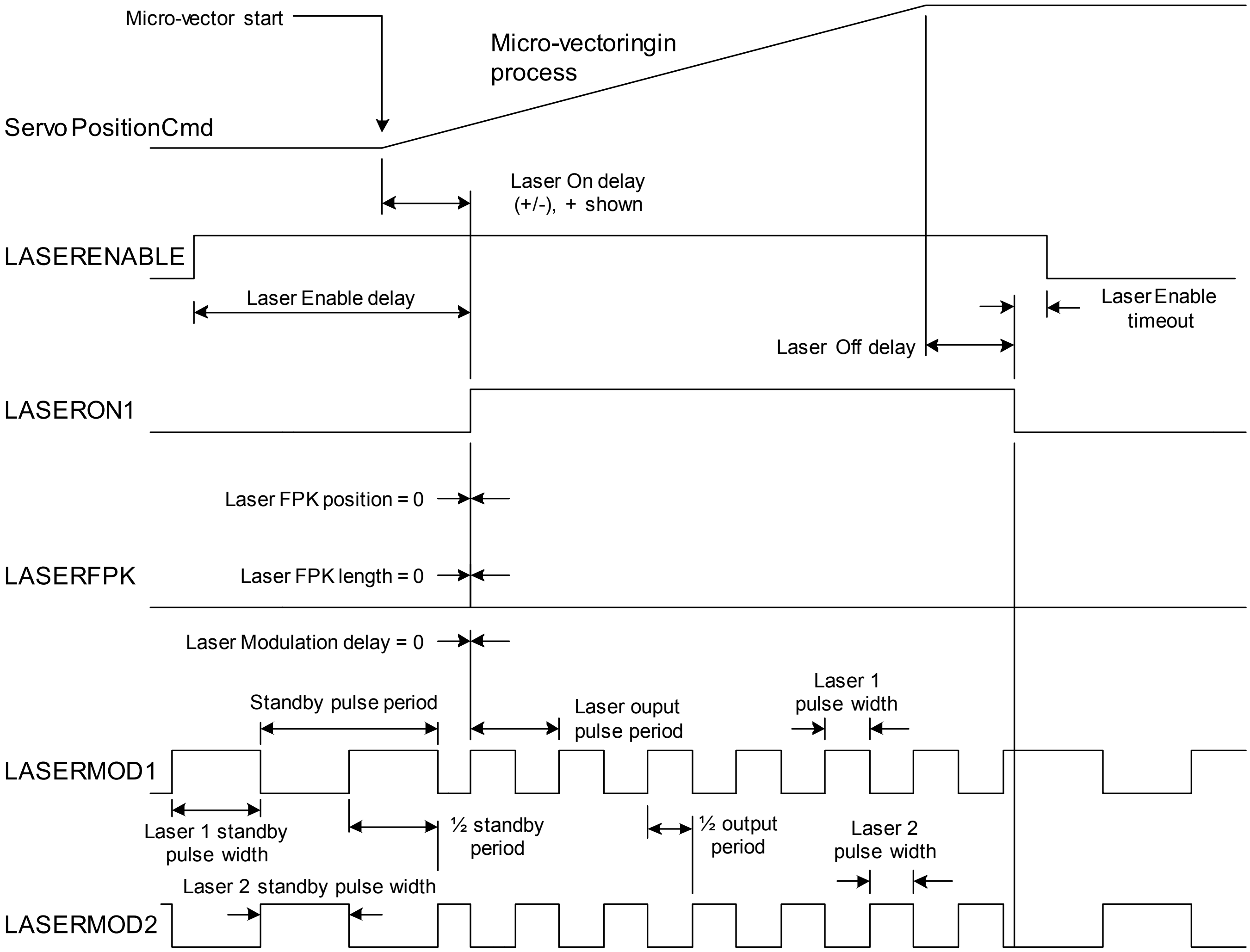


Figure 44 Fiber Laser Timing

Pulsed fiber lasers have recently become very popular because of a reduced cost of ownership relative to more traditional YAG lasers. The IPG YLP series of lasers introduces a new control signal requirement that is met with the LASERENABLE signal of the EC1000. The MO (Master Oscillator) signal defined in the IPG "B" interface specification is intended to be driven by the Laser Enable signal of the EC1000. This signal is used to prepare the fiber laser to generate ouput pulses and must be asserted at least 7ms before pulses are required. In addition, this signal should be deasserted after laser emission in order to save power and extend laser life-time. Deassertion, however, should not be done too quickly in order to avoid the overhead of restarting the laser. Deassertion is usually done after all marking is done in a job. In the case of the EC1000, a timeout is provided to automatically deassert the LASERENABLE signal after a period of inactivity.

In the above diagram notice that the LASERFPK signal is made inactive, i.e. it is not required by the interface. The pulse width of the standby and active periods is set to 50% of the pulse period (square wave) since laser emission is triggered on the leading edge of the pulse. Pulse width does not determine the level of power emitted, only the pulse frequency (or period) determines average power. In practice, the pulse width to period ratio can be in a range of 0.1 to 0.9.

⚠ CAUTION⚠

The IPG laser specifies that the pulse period must not be longer than a minimum value. The EC1000 does not protect against incorrect programming; the application must prevent incorrect values from being used.

The IPG laser as a GUIDELASER signal to turn a pointer laser on/off. This signal can be controlled directly with the LASERON2 signal if it is configured correctly (see the example below). In addition, a DLATCH signal is required to latch the laser digital power value. A special mode of operation of the EC1000 laser digital output port can support this feature, albeit at the sacrifice of the least significant bit of laser data. This configuration is specified below.

Table 9: Example IPG Fiber Laser Configuration XML

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<LsrEnaDly>**7000**</LsrEnaDly>	<set id=”LaserEnableDelay”>**7000**</set>	Minimum master oscillator startup time
<LsrEnaTmo>**10000**</LsrEnaTmo>	<set id=”LaserEnableTimeout”>**10000**</set>	Shut down laser master oscilator if no laser activity for 10msec
<LsrModDly>**0**</LsrModDly>	<set id=”LaserModDelay”>**0**</set>	No modulation delay required
<FpsPos>**0**</FpsPos>	<set id=”LaserFPK”>**0, 0**</set>	No FPK required
<FpsWidth>**0**</FpsWidth>		
<TickleWidth1>**25**</TickleWidth1>	<set id=”LaserStandby”>**1, 25, 50**</set>	Laser 1 stand-by; pulse width == 25 laser timing ticks (25µsec); pulse period == 50 ticks (50µsec) == 20.0KHz
<TickleFreq1>**20**</TickleFreq1>		
<TickleWidth2>**25**</TickleWidth2>	<set id=”LaserStandby”>**1, 25, 50**</set>	Laser 2; Settings the same as Laser 1
<TickleFreq2>**20**</TickleFreq2>		
N/A	<set id=”LaserOnDelay”>**150**</set>	150 laser timing ticks == 150µsec
N/A	<set id=”LaserOffDelay”>**100**</set>	100 laser timing ticks == 100µsec
N/A	<set id=”LaserPulse”>**1, 5, 10**</set>	Laser 1 operating; pulse width == 5 laser timing ticks (5µsec); pulse period == 10 ticks (10µsec) == 100.0KHz
N/A	<set id=”LaserPulse”>**2, 5, 10**</set>	Laser 2 operating; pulse width == 5 laser timing ticks (5µsec); pulse period == 10 ticks (10µsec) == 100.0KHz, must be same as Laser 1
<LON2Cfg>1</LON2Cfg>	N/A	Sets the mode of LASERON2 to be asserted when LASERON1 would be asserted, but only if the laser is disabled.
<LsrPwrMode>7bit</LsrPwrMode>	N/A	Set the configuration of the laser digital power port so the bit 0 can be tied to the DLATCH signal. This bit will toggle 0->1->0 after each power change.

6 Application Programming Interface

The host software Application Programming Interface (API) is supplied as Windows .NET assemblies and COM objects that can be accessed from any suitable Microsoft Windows platform programming language, such as Visual Basic, C++, C#, etc. For convenience, the API is defined using Visual Basic syntax.

The API makes extensive use of XML to pass parameters between a client application and the library. This technique dramatically reduces the number of interface methods required to control an EC1000 module. The sections on Data Definitions explicitly defines the XML interface requirements.

The API is divided into two components: the Broadcast API is used to identify EC1000 modules on the network and the Session API is used to transfer configuration and job data to and from a selected controller.

6.1 API Implementation and Installation

The API is implemented in Microsoft's C# language and is exposed as Windows .NET assemblies and as COM objects. The DLLs and .tlb files that make up the interface are automatically installed and registered in the Window Registry by a setup installation program on the software distribution CD.   Installation details are in Appendix B.

In Visual Studio Version 6 programming languages, the API is accessed as COM objects that are imported into the IDE trough the use of the COM object browser. The interfaces are identified as ILECBroadcast and ILECSession. In languages based on Microsoft .NET technology, the interfaces are available as assemblies that can be referenced within a project. Example code that illustrates the use of the API is on the distribution CD and installed on the computer during API installation. The code examples are in a set of subdirectories in the Sample Programs directory where the API software is installed.

6.2 Broadcast API

The Broadcast API is a set of methods that allow a client application to identify EC1000 controllers on the network and to get relevant information about those controllers. On a configurable periodic basis, the EC1000 modules broadcast identification packets out onto the network. The API captures broadcast messages from all available EC1000 controllers and makes this information available to the client. This information is used by the client to establish a communication session with a target controller. Sessions are used to send job data to a controller, and to send/receive module configuration data. The methods used in sessions are described in the Session API section.

6.2.1 Attach Broadcast

Command	ILecBroadcast.clientAttachBroadcast
Purpose	Establish a connection to receive broadcast messages
Usage	ILecBroadcast.clientAttachBroadcast (ByVal pstrMulticastAddress As String, // IP address to which the EC devices are broadcasting over (224.168.100.2) ByVal pstrLocalAddress As String, // IP address of the local network adaptor that is connected to the EC1000 // modules. ByVal piLocalPortNumber As Long, // Port number to which the EC devices are broadcasting over (11000) ByRef piClientId As Long // Identifier of the successful connection made by the application) As Unsigned Long
Explanation	This method is used by a client application to establish a connection to the broadcast mechanism of the EC1000. Once connected, a client may receive broadcast messages from all EC1000 module on the network. The messages contain information about the broadcasting module including the name, internet IP address, and other relevant data. This data is retrieved through the use of Broadcast.GetBroadcastData(). pstrAddress and piPortNumber are values that are defined in the AdminConfig file (see section: Administration Configuration) pstrLocalAddress is required to differentiate which network adaptor is connected to the EC1000. The source code for a sample utility function to get this information from the Windows operating system is provided in the Sample Programs\LecTesterUtils directory.
Returns	0 - Success 1 - Could not establish a connection
See also	ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerCount(), ILecBroadcast.getLecServerList(), ILecBroadcast.getBroadcastData()

6.2.2 Detach Broadcast

Command	ILecBroadcast.clientDetachBroadcast
Purpose	Terminate the connection to the broadcast mechanism
Usage	ILecBroadcast.clientDetachBroadcast (ByVal piClientId As Long // Identifier of the connection made by the application) As Unsigned Long
Explanation	This method is used by a client application to terminate a connection to the broadcast mechanism of the EC1000.
Returns	0 - Success
See also	ILecBroadcast.clientAttachBroadcast()

6.2.3 Get Lec Server Count

Command	ILecBroadcast.getLecServerCount
Purpose	Get embedded controller device data
Usage	ILecBroadcast.getLecServerCount(ByVal piClientId As Long, // Identifier of the connection made by the application ByRef piServerCount As Long, // The number of EC1000 devices that were identified) As Unsigned Long
Explanation	Once a connection to the broadcast mechanism has been established, broadcast messages are then received and a table of available modules is built by the API. This method returns the number of distinct EC1000 modules that have transmitted valid broadcast packets since the Broadcast.clientAttachBroadcast() method was called. Because of the asynchronous and periodic nature of the broadcast transmissions, it may take some time before all EC1000 controllers are recognized and reported via this method. Several successive calls may yield different results until enough time has passed to account for the longest broadcast interval. The broadcast interval is configured using the Session.requestFixedData() and Session.setFixedData() methods with the AdminConfig data as an argument.
Returns	0 – Success 4 – Illegal client identifier
See also	ILecBroadcast.clientAttachBroadcast(), ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerList(), ILecBroadcast.getBroadcastData()

6.2.4 Get Lec Server List

Command	ILecBroadcast.getLecServerList
Purpose	Get embedded controller device data
Usage	ILecBroadcast.getLecServerList(ByVal piClientId As Long, // Identifier of the connection made by the application ByRef piServerCount As Long, // The number of EC1000 devices that were identified ByRef pstrFriendlyName As String // The names of the EC1000 devices that were identified. The string returned // contains an XML representation of the data.) As Unsigned Long
Explanation	This method returns a list of identifiers for the EC1000 modules for which valid broadcast packets have been received. One of the friendly names can used in the method Broadcast.getBroadcastData() to obtain more extensive identification data. Because of the asynchronous and periodic nature of the broadcast transmissions, it may take some time before all EC1000 controllers are recognized and reported via this method. Several successive calls may yield different results until enough time has passed to account for the longest broadcast interval. The broadcast interval is configured using the Session.requestFixedData() and Session.setFixedData() methods with the AdminConfig data as an argument.
Returns	0 – Success 4 – Illegal client identifier The friendly name list contains an XML representation of the data. For example: <LecList> <Lec name=”EC_Alpha” ip=”192.168.42.30” mac=”00:50:C2:4F:A0:01” /> <Lec name=”EC_Beta” ip=”192.168.42.31” mac=”00:50:C2:4F:A0:06” /> </LecList>
See also	ILecBroadcast.clientAttachBroadcast(), ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerCount(), ILecBroadcast.getBroadcastData

6.2.5 Get Broadcast Data

Command	ILecBroadcast.getBroadcastData
Purpose	Get embedded controller device data
Usage	ILecBroadcast.getBroadcastData(ByVal piClientId As Long, // Identifier of the connection made by the application ByVal pstrFriendlyName As String, // Name of the EC device ByVal piDataType As Long, // The type of EC device data (see Broadcast Data Definitions section) ByRef piData As String // The data requested from the EC device. The string returned contains an // XML representation of the data requested by piDataType) As Unsigned Long
Explanation	This function is used by a client application to retrieve various types of data related to the specified EC1000 module. This data is defined in the Data Types section.
Returns	0 - Success 4 – Illegal client identifier 8 – Server name not found
See also	ILecBroadcast.clientAttachBroadcast(), ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerCount(), ILecBroadcast.getLecServerList()

Broadcast Data Definitions

The both the Broadcast and Session APIs uses a data type code to specify the data that the application is requesting or sending. This is the piDataType argument in the methods Broadcast.getBroadcastData(), Session.requestFixedData(), and Session.sendFixedData(). All data types support an XML representation of the data.

Broadcast Data Type	piDataType Value Code
System Information	0x01
Status Information	0x07

In the following data description tables, example data is shown in **bold** font. Although in XML all data is expressed as text, the actual data type interpretation is application dependent. For the EC1000, all data has an expected type interpretation, thus the tables contain a collumn that indicates the data type that is intended for the particular data element. The data types are identified as follows:

Type Identifier	Type Description	Range
STR	ASCII String	<= 256 characters
U16	Unsigned 16-bit Integer	0 <-> 65535
I16	Signed 16-bit Integer	-32768 <-> +32767
U32	Unsigned 32-bit Integer	0 <-> 4,294,967,295
I32	Signed 32-bit Integer	-2,147,483,648 <-> 2,147,483,647
FLT	Floating point	IEEE 64-bit Floating Point range
BOOL	Boolean	true, false

All the data retrievable using the Broadcast.getBroadcastData method is read-only.

System Information

<table>
<tr><td>Purpose</td><td colspan="4">The system information data contains device, hardware, and connection information</td></tr>
<tr><td rowspan="25">Definition</td><td>XML Tag</td><td>XML Example Text</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td>&lt;Data type=”SysInfoData” rev=”1.0”&gt;</td><td></td><td></td></tr>
<tr><td>MSN</td><td>&lt;MSN&gt;<b>EC1000-000005</b>&lt;/MSN&gt;</td><td>STR</td><td>Unique board manufacturing code.</td></tr>
<tr><td>PVer</td><td>&lt;PVer&gt;<b>0420</b>&lt;/PVer&gt;</td><td>STR</td><td>Version of the platform software.</td></tr>
<tr><td>AVer</td><td>&lt;AVer&gt;<b>1.0.0</b>&lt;/AVer&gt;</td><td>STR</td><td>Version of the EC1000 embedded server software.</td></tr>
<tr><td>ObjExtVer</td><td>&lt;ObjExtVer&gt;<b>cti.1.0</b>&lt;/ObjExtVer&gt;</td><td>STR</td><td>Version of the on-board object extension library</td></tr>
<tr><td>FPGAFirmVer</td><td>&lt;FPGAFirmVer&gt;<b>20060131</b>&lt;/FPGAFirmVer&gt;</td><td>STR</td><td>Version of the FPGA firmware that is loaded.</td></tr>
<tr><td>StateCode</td><td>&lt;StateCode&gt;<b>1</b>&lt;/StateCode&gt;</td><td>U32</td><td>Connection status of EC1000. The state-codes are:
<table>
<tr><th>State</th><th>Value</th><th>Meaning</th></tr>
<tr><td>Available</td><td>0</td><td>Available for connection</td></tr>
<tr><td>ClientTCP</td><td>1</td><td>Connected to network client</td></tr>
<tr><td>ClientSerial</td><td>2</td><td>Connected to serial client <i>(future)</i></td></tr>
<tr><td>ClientLocal</td><td>4</td><td>In local mode</td></tr>
<tr><td>Restarting</td><td>8</td><td>Server restarting</td></tr>
<tr><td>Waiting</td><td>16</td><td>Waiting for server startup</td></tr>
<tr><td>Pausing</td><td>32</td><td>Job paused</td></tr>
<tr><td>WaitingTCP</td><td>64</td><td>Waiting for TCP connection</td></tr>
<tr><td>NotAvailable</td><td>128</td><td>Server is in a transitional state and unavailable</td></tr>
</table></td></tr>
<tr><td>LastError</td><td>&lt;LastError&gt;<b>0</b>&lt;/LastError&gt;</td><td>I32</td><td>Last system error. For instance, 9001 represents a recent abort operation had completed.</td></tr>
<tr><td>FreeTempStorage</td><td>&lt;FreeTempStorage&gt;<b>7805</b>&lt;/FreeTempStorage&gt;</td><td>U32</td><td>The amount of free storage in non-persistent memory in Kilo Bytes.</td></tr>
<tr><td>PermStoragePath</td><td>&lt;PermStoragePath&gt;<b>Disk</b>&lt;/PermStoragePath&gt;</td><td>STR</td><td>The path to the root of persistent memory.</td></tr>
<tr><td>FreePermStorage</td><td>&lt;FreePermStorage&gt;<b>29616</b>&lt;/FreePermStorage&gt;</td><td>U32</td><td>The amount of free storage in persistent memory in Kbytes.</td></tr>
<tr><td>FreeUSBStorage</td><td>&lt;FreeUSBStorage&gt;<b>100220</b>&lt;/FreeUSBStorage&gt;</td><td>U32</td><td>The amount of free storage in Kbytes on the USB Flash device if present on the system.</td></tr>
<tr><td>MAC</td><td>&lt;MAC&gt;<b>00:50:C2:4F:A0:00</b>&lt;/MAC&gt;</td><td>STR</td><td>Hardware address.</td></tr>
<tr><td>NetMask</td><td>&lt;NetMask&gt;<b>255.255.255.0</b>&lt;/NetMask&gt;</td><td>STR</td><td>Network mask used by EC1000. This value is either manually set, or provided by a DHCP or DNS server.</td></tr>
<tr><td>NetAssign</td><td>&lt;NetAssign&gt;<b>1</b>&lt;/NetAssign&gt;</td><td>I32</td><td>Network assignment is either manual, provided by DHCP, or provided by DNS.</td></tr>
<tr><td>IP</td><td>&lt;IP&gt;<b>192.168.2.1</b>&lt;/IP&gt;</td><td>STR</td><td>IP address used by EC1000. This value is either manually set, or provided by a DHCP or DNS server. This IP address is used in the Session.loginSession method to connect to a specific EC1000.</td></tr>
<tr><td>ConnectIP</td><td>&lt;ConnectIP&gt;<b>192.168.2.65</b>&lt;/ConnectIP&gt;</td><td>STR</td><td>The client IP address that is currently connected to EC1000.</td></tr>
<tr><td>FriendlyName</td><td>&lt;FriendlyName&gt;<b>EC_Alpha</b>&lt;/FriendlyName&gt;</td><td>STR</td><td>Name used by EC1000.</td></tr>
<tr><td>ConnectJob</td><td>&lt;ConnectJob&gt;<b>Hubble</b>&lt;/ConnectJob&gt;</td><td>STR</td><td>The job name that is currently marking.</td></tr>
<tr><td>Port</td><td>&lt;Port&gt;<b>12200</b>&lt;/Port&gt;</td><td>U32</td><td>The network port currently in use by the Job Session.</td></tr>
<tr><td>HSN</td><td>&lt;HSN&gt;<b>HEAD-0000023</b>&lt;/HSN&gt;</td><td>STR</td><td>Marking head serial number.</td></tr>
<tr><td></td><td>&lt;/Data&gt;</td><td></td><td></td></tr>
<tr><td>Explanation</td><td colspan="4">This data define the basic characteristics of the controller, especially that required to properly communicate with the controller. It contains a combination of live dynamic data, and static data that is stored on the Flash memory of the device. All data is read-only.</td></tr>
<tr><td>See also</td><td colspan="4">ILecBroadcast.getBroadcastData()</td></tr>
</table>

Status Information

Purpose	The status information data contains the current status maintained by the marking engine			
Definition	XML Tag	XML Example Text	Type	Description
	Data	<Data type=”StatInfoData” rev=”1.0”>		StatInfoData identifier
	XPosAck	<XPosAck>**true**</XPosAck>	BOOL	Boolean passed from the X axis galvo servo controller indicating that the servo is “settled” at the commanded position. Note that this feature is not supported by all galvo controllers.
	YPosAck	<YPosAck>**true**</YPosAck>	BOOL	Boolean passed from the Y axis galvo servo controller indicating that the servo is “settled” at the commanded position. Note that this feature is not supported by all galvo controllers
	XPos	<XPos>**-2489**</XPos>	I16	The value of the current ideal commanded X position prior to lens correction.
	YPos	<YPos>**5510**</YPos>	I16	The value of the current ideal commanded Y position prior to lens correction.
	XActPos	<XActPos>**-2489**</XActPos>	I16	The value of the actual X position after lens correction.
	YActPos	<YActPos>**5510**</YActPos>	I16	The value of the actual Y position after lens correction.
	XTemp	<XTemp>**29.3**</XTemp>	FLT	The value of the temperature in Celsius of the X servo. Note that this feature is not supported by all galvo controllers.
	YTemp	<YTemp>**28.5**</YTemp>	FLT	The value of the temperature in Celsius of the Y servo. Note that this feature is not supported by all galvo controllers.
	ContrlTemp	<ContrlTemp>**32.0**</ContrlTemp>	FLT	The value of the temperature in Celsius of the EC1000 controller.
	XPower	<XPower>**true**</XPower>	BOOL	Boolean determining if the X servo is powered and ready. Note that this feature is not supported by all galvo controllers.
	YPower	<YPower>**true**</YPower>	BOOL	Boolean determining if the Y servo is powered and ready. Note that this feature is not supported by all galvo controllers.
	Interlock	<Interlock>**4**</Interlock>	U16	This number represents a bitmask that encodes the current state of the system interlock switches. A “1” in the bit position means that the interlock has been broken in that position. Bits[3..0] represent the state of the signals INTERLOCK[4..1]
	CurrentDIO	<CurrentDIO>**0x1023**</CurrentDIO>	U16	This number represents a bitmask that encodes the current state of the system digital I/O lines. bits[3..0] == USERIN[4..1] bit[5..4] == SPAREIN, STRTMRK bits[9..6] == INTERLOCK[4..1] bits[13..10] == USEROUT4..1] bits[17..14] == SPAREOUT, LEC_ERROR, LEC_BUSY, MRKINPRG
	JobMarker	<Jobmarker>**35**</JobMarker>	U16	This number is a copy of the current job marker data register that can be set by an application job via the JobMarker instruction.
	JobDataCntr	<JobDataCntr>**32336**</JobDataCntr>	U32	This number is a copy of the current job data counter. This counter is cleared whenever the marking engine enounters a StartJob instruction and can also be initialized by an application job via the JobDataCounter instruction. This counter represents the number of 32-bit data elements that the marking engine has processed since the last time this value was reset.
		</Data>		End StatInfoData
Explanation	This data represents the live status of the device. All data is read-only.			
See also	ILecBroadcast.getBroadcastData()			

6.3 Session API

Once all EC1000 controllers are identified using the Broadcast API, individual controllers may be selected for subsequent communication. The Session API provides the methods to connect to a target EC1000, to get and set configuration data, to send job data, and to manage asynchronous communcations events generated by the controller.

6.3.1 Session Login

Session are established via a login operation to the target E1000.

Command	ILecSession.loginSession
Purpose	Connect to an EC device by establishing a session

Usage	ILecSession.loginSession(
	ByVal pstrLocalAddress As String, // IP address of the local network adaptor that is connected to the EC1000
	// modules.
	ByVal pstrAddress As String, // TCP/IP Address of the EC1000 to login. This is the "ip" attribute of the
	// EC1000 selected by the application and identified in the
	// Broadcast.getLecServerList data
	ByVal piPortNumber As Long, // Network Port on the EC1000 supporting the session. This is the <Port>
	// value of the SysInfoData returned from the Broadcast.getBroadcastData call
	// for the selected EC1000
	ByVal pstrUsername As String, // Reserved for future use
	ByVal pstrPassword // Reserved for future use
	ByVal piTimeout As Unsigned Long // Duration for attempting call in seconds
	) As Unsigned Long

Explanation	Once EC1000 modules have been identified via the use of Broadcast API, a communications session can be opened between the client and a selected target EC1000. Sessions are established via a call to this method. Multiple sessions to *different* target EC1000 controllers are made by instantiating separate Session objects. A target EC1000 controller may only serve one client session at a time. pstrLocalAddress is required to differentiate which network adaptor is connected to the EC1000. The source code for a sample utility function to get this information from the Windows operating system is provided in the Sample Programs\LecTesterUtils directory.
Returns	0 – Success 2 – Could not establish a connection 16 – Could not start event handling 18 – Cancelled by user 24 - A timeout occured waiting for the operation to complete
See also	ILecSession.logoutSession(), ILecSession.requestFixedData(), ILecSession.sendFixedData(), ILecSession.sendSteamData(), ILecSession.sendPriorityData()

6.3.2 Session Logout

Sessions with a connected E1000 are terminated via a logout operation.

Command	ILecSession.logoutSession
Purpose	Disconnect an EC device session
Usage	ILecSession.logoutSession(ByVal puiTimeout As Unsigned Long // Duration for attempting call in seconds) As Unsigned Long
Explanation	When session communication is completed, the client closes the session via a call to this method. Once the session is closed, another new session may be opened to the same or other EC1000 devices via a call to Session.loginSession(). Note that if a job was streamed out to the EC1000 and was still executing when the logout was invoked, the job will be immediatley aborted.
Returns	0 – Success
See also	ILecSession.loginSession()

6.3.3 Session Request Fixed Data

The EC1000 has the ability to store a large amount of data in non-volatile Flash memory. This data can be configuration data or job data. Configuration data is classified a "fixed" data, i.e. it has a lifetime that spans boot-up cycles of the controller. Some of the configuration data is set at the factory and is considered permanent read-only information. Other data is used by the controller at boot-up to properly initialize the hardware interfaces, and still other data is provided for the convenience of the application programmer to indicate the capabilities of the integrated system. All data is defined in the Session Fixed Data Definitions section of this manual.

The fixed data stored on the EC1000 is accesed by requesting it from the controller..

Command	ILecSession.requestFixedData
Purpose	Retrieve fixed data from an EC device session

Usage	ILecSession.requestFixedData(
	ByVal piDataType As Long Identifier of the requesting data. See Session Fixed Data Definitions section.
	ByVal pstrStorageName As String // File *name* of the data file. The file *path* is constructed by the API as follows:
	// <PermStoragePath>\LEC\Config\<pstrStorageName>.xml
	// where <PermStoragePath> is defined in the SysInfoData for the selected
	// EC1000 and <pstrStorageName> is the name of the selected fixed data file
	// as stored on the EC1000 without the ".xml" extension.
	ByRef pstrData As String, // Requested data
	ByVal puiTimeout // Duration for attempting call in seconds
	) As Unsigned Long

Explanation	EC1000 modules are autonomous devices that contain information that configures the module at boot-up for the particular hardware arrangement of the marking head. This information defines such things as the laser interface, the lens characteristics, and the optical system correction tables. An application can access this information by specifying the data type using the piDataType argument and providing a file name for the data as stored on the EC1000. The information is returned as an XML string which must be decoded by the application. The XML specification for the different data types is defined in the Session Fixed Data Definitions section. The AdminConfig.xml data file (see section Administration Configuration) contains an element definition <ControlFile> naming the master EC1000 controller configuration file. Within this file are element definitions naming the currently active lens, laser, correction table, and user definitions files. These names are typically used as the <pstrStorageName> argument above, although other files may be accessed on the EC1000 file system if those file names are known and the files are of the proper type.
Returns	0 – Success 10 – Network connection not established 11 – Requested data not found on EC device 12 – Location of data not found on EC device 13 – Cannot access the remote location where the data resides on the EC device 14 – Cannot access the local destination location where the data is written to 15 – The data type requested is unknown 24 - A timeout occured waiting for the operation to complete
See also	ILecSession.sendFixedData()

Session Fixed Data Definitions

The Session API uses a data type code to specify the data that the application is requesting or sending. This is the piDataType argument in the methods Session.requestFixedData(), and Session.sendFixedData(). All data types support an XML representation of the data.

Fixed Data Type	Data ID
Controller Configuration	0x05
Laser Configuration	0x06
Lens Configuration	0x02
Correction Table	0x0D
User Configuration	0x0F
Performance Adjustments	0x10
Admin Configuration	0x0A

In the following data description tables, example data is shown in **bold** font. Although in XML all data is expressed as text, the actual data type interpretation is application dependent. For the EC1000, all data has an expected type interpretation, thus the tables contain a collumn that indicates the data type that is intended for the particular data element. The data types are identified as follows:

Type Identifier	Type Description	Range
STR	ASCII String	<= 256 characters
U16	Unsigned 16-bit Integer	0 <-> 65535
I16	Signed 16-bit Integer	-32768 <-> +32767
U32	Unsigned 32-bit Integer	0 <-> 4,294,967,295
I32	Signed 32-bit Integer	-2,147,483,648 <-> 2,147,483,647
FLT	Floating point	IEEE 64-bit Floating Point range
BOOL	Boolean	true, false
HEX	Unsigned 16-bit integer	0x0000 <-> 0xFFFF

All data that can be retrieved with the Session.requestFixedData() method is changable with the Session.sendFixedData() method. This powerful interface permits full configurability of the EC1000 controller. Most of the elements in the data tables are set by a system integrator to provide information for a marking application programmer to configure the user-interface and control interfaces as a function of the controller/system configuration. This data is not intendend to be changed after it has been set by an integrator.

In addition to the integrator data, there is a table of data that is intended to be set by a system administrator. This data can be adapted at the end-customer site to meet specific networking requirements. This data is also intended to be read-only from a marking application perspective.

Most of the properties defined in the configuration data tables are provided as a convenience to the application programmer in adapting the software for various target configurations. These properties are flagged with the letter "A" in the *Use* column. Some of the data is used by the controller at boot-up to configure the laser control signals. This data is flagged with the letter "C" in the *Use* column. All of the data is persistent on the controller and changeable via the API.

Administration Configuration

Administration Configuration data defines the base behavior of the module. Most of the items defined here are used to configure the network parameters and diagnostic tracing of the server software. The <ControlFile> tag is important in that it defines the name of the controller configuration file which contains pointers to other files that define the configuration of the lens and laser among other things.

Controller Configuration

The Controller Configuration file is the master control file for defining the startup configuration of the controller. It contains pointers to other configuration files that deal with specific elements of the system such laser timing, correction tables, lens identification, user adjustments, etc. The file names referenced in the table are XML file names with the .xml extension suppressed. The files are in the <PermStoragePath>\LEC\Config directory on the EC1000. PermStoragePath is the value reported in the broadcasted SystemData packets.

Purpose	The controller configuration describes to the combination of components that make up the EC device				
	XML Tag	XML Example Text	Type	Use	Description
Definition	Data	<Data type="ControlConfigData" rev="1.0">			ControlConfigData identifier
	CorrFile1	<CorrFile1>**CORRTAB1**</CorrFile1>	STR	C	The name of correction table 1 file
	CorrFile2	<CorrFile2>**CORRTAB2**</CorrFile2>	STR	C	The name of correction table 2 file
	LensFile	<LensFile>**LENSFILE2**</LensFile>	STR	C	The name of the lens configuration file
	LaserFile	<LaserFile>**LASERFILE4**</LaserFile>	STR	C	The name of the laser configuration file
	UserFile	<UserFile>**MYCONFIGFILE**</UserFile>	STR	C	The name of the user configuration file
	PerformanceFile	<PerformanceFile>**PADJUST**</ PerformanceFile>	STR	C	The name of the performance adjustments file
	MotfCapable	<MotfCapable>**true**</MotfCapable>	BOOL	A	System is Mark On The Fly (MOTF) capable (true)
	MotfEncoderCal	<MotfEncoderCal>**24.23**</ MotfEncoderCal>	FLT	C	MOTF calibration factor. Relates the encoder counts to laser positioning bits (bits/count)
	MotfCalGain	<MotfCalGain>**1.0**</MotfCalGain>	FLT	A	MOTF digital gain factor. Used as a fine-tuning scalar adjustment of MotfEncoderCal.
	MotfMode	<MotfMode>**0**</MotfMode>	U16	C	MOTF operational mode 0 - Use encoder 1 - Simulate endoder
	MotfDirection	<MotfDirection>**0**</MotfDirection>	I16	C	MOTF orientation and direction in degrees 0 - left to right in the X axis 90 - bottom to top in the Y axis 180 - right to left in the X axis 270 - Top to bottom in the Y axis
	AxisDACRange	<AxisDACRange>0x**1**</AxisDACRange>	HEX	C	Bit-field encoded values that set the output votage range of the X and Y DACs and the Z DAC as follows: Bits [1..0] encode the X and Y axis Bits [3..2] encode the Z axis 0 - ±2.5V single ended, 5V differential 1 - ±5V single ended, 10V differential 2 - ±10V single ended, 20V differential
	LaserPipelineDelay	<LaserPipelineDelay>**1500**</ LaserPipelineDelay>	U16	C	The time in laser timing ticks that all laser signals are delayed relative to micro-vector generation
	IntlockConfig	<IntlockConfig>**0x1707**</IntlockConfig>	HEX	C	Interlock configuration control. There are two fields in the argument: Polarity and Enable: Polarity Bits [3..0] represent the interlock signals INTLOCK[4..1]. A "1" corresponds no current flowing through the interlock optical isolator being the "asserted" state. Enable Bits [11..8] represent the interlock signals INTLOCK[4..1]. A "1" enables a transition of the interlock signal going from the unasserted to the asserted state togenerate an "Interlock" exception and shut down an active job provided that bit 12 is also asserted. Bit [12] is the master enable bit for the interlock function. If this bit is set, then all enabled interlock signals should be de-asserted at power-up time or else an immediate "Interlock" exception will be generated when this parameter is processed. All of the Enable bits can also be manipulated using the SetInterlockEnable priority data message. If an interlock that is enabled is tripped, the condition that caused the trip must be cleared and an "Abort" priority message sent before a job can be restarted without generating another "Interlock" exception. The current state of the interlock physical signals can be seen in the Broadcast Status data as element <Interlock>

<table>
<tr><td rowspan="2">Purpose</td><td colspan="5">The controller configuration describes to the combination of components that make up the EC device</td></tr>
<tr><td>XML Tag</td><td>XML Example Text</td><td>Type</td><td>Use</td><td>Description</td></tr>
<tr><td rowspan="4">Definition</td><td>ServoConfig</td><td><ServoConfig><b>0x4</b><ServoConfig>
.</td><td>HEX</td><td>C</td><td>Servo configuration control. The value is bit-field encoded as follows:
<table>
<tr><th>Name</th><th>Value</th><th>Definition</th></tr>
<tr><td>X&Y SERVO_EN polarity</td><td>0x0001</td><td>0=active high, 1=active low</td></tr>
<tr><td>Z SERVO_EN polarity</td><td>0x0002</td><td>0=active high, 1=active low</td></tr>
<tr><td>Enable X, Y Servos</td><td>0x0004</td><td>1=enable servos, 0=disable</td></tr>
<tr><td>Enable Z Servo</td><td>0x0008</td><td>1=enable servos, 0=disable</td></tr>
<tr><td>X&Y SERVO_RDY polarity</td><td>0x0010</td><td>0=active high, 1=active low</td></tr>
<tr><td>Z_SERVO_RDY polarity</td><td>0x0020</td><td>0=active high, 1=active low</td></tr>
<tr><td>X, Y Not-ready exception enable</td><td>0x0040</td><td>1=enable exception event generation is X or Y servo becomes not ready</td></tr>
<tr><td>Z Not-ready exception enable</td><td>0x0080</td><td>1=enable exception event generation if Z servo becomes not ready</td></tr>
</table></td></tr>
<tr><td>JumpPeriodOffset</td><td><JumpPeriodOffset><b>0</b></JumpPeriodOffset></td><td>I32</td><td>C</td><td><i>Reserved for future use.</i></td></tr>
<tr><td>MarkPeriodOffset</td><td><MarkPeriodOffset><b>0</b></MarkPeriodOffset></td><td>I32</td><td>C</td><td><i>Reserved for future use.</i></td></tr>
<tr><td></td><td></Data></td><td></td><td></td><td>End ControlConfigData</td></tr>
<tr><td>Explanation</td><td colspan="5">These values are normally set by the integrator and not intended to be altered by a marking application.
Note that when the Controller Configuration is sent to EC1000, the correction table and laser configurations referenced are also applied to the controller. Consequently, whenever these configurations are updated, the current correction table is always reset to CorrFile1.
Detailed Motf operation is controlled through instructions passed as part of the job stream and is not a “mode” of the controller.</td></tr>
<tr><td>See Also</td><td colspan="5">ILecSession.requestFixedData(), ILecSession.sendFixedData()</td></tr>
</table>

Laser Configuration

Purpose	The laser configuration defines the properties of the laser in use with the EC device				
	XML Tag	XML Example Text	Type	Use	Description
Definition	Data	<Data type=”LaserConfigData” rev=”1.0”>			LaserConfigData identifier
	LsrName	<LsrName>**IPC002**</LsrName>	STR	A	The name of the laser
	FixedFreq	<FixedFreq>**true**</FixedFreq>	BOOL	A	Laser is only capable of a fixed frequency setting (true), or capable of variable frequency settings (false)
	FixedPW	<FixedPW>**true**</FixedPW>	BOOL	A	Laser is only capable of a fixed pulse width setting (true), or capable of variable pulse width settings (false)
	FixedWatts	<FixedWatts>**true**</FixedWatts>	BOOL	A	Laser is only capable of a fixed ouput power setting (true), or capable of variable ouput power settings (false)
	WattsUnits	<WattsUnits>**true**</WattsUnits>	BOOL	A	Laser power units are in Watts (true), or % duty-cycle (false)
	Pulse	<Pulse min=”**2**” max=”**65535**”/>	U16	A	pulse width range supported by the laser (µsec)
	Bits	<Bits min=”**0**” max=”**255**”/>	U16	A	Binary value range for lasers with digital power control
	ExtPwrCtrl	<ExtPwrCtrl>**false**</ExtPwrCtrl>	BOOL	A	Laser power is controllable via an external knob (true)
	UseExtPwrCtrl	<UseExtPwrCtrl>**false**</UseExtPwrCtrl>	BOOL	A	Application is configured to use external power control (true)
	VisPtr	<VisPtr>**false**</VisPtr>	BOOL	A	Laser has a visible pointer integrated into it (true)
	Duty	<Duty min=”**2**” max=”**90**”/>	U16	A	Duty cycle range of the laser pulses (%)
	LsrType	<LsrType>**1**</LsrType>	U16	A	Application definable value to identify a laser type
	Freq	<Freq min=”**2**” max=”**100**”/>	U16	A	Pulse frequency range sustainable by the laser (KHz)
	Watts	<Watts min=”**1**” max=”**15**”/>	U16	A	Wattage range producible by the laser
	Volts	<Volts min=”**1**” max=”**10**”/>	U16	A	Analog power level voltage range sustainable by the laser. The EC1000 is capable of 0-10 Volts output.
	Interlock	<Interlock>**IPCIntlocks.txt**</Interlock>	STR	A	The name of a file on the host platform that contains instructions on how to clear an interlock break
	LENAHigh	<LENAHigh>**false**</LENAHigh>	BOOL	C	LASERENABLE signal is active high (true) or active low (false)
	LONHigh	<LONHigh>**false**</LONHigh>	BOOL	C	LASERON1 signal is active high (true) or active low (false)
	LON2High	<LON2High>**false**</LON2High>	BOOL	C	LASERON2 signal is active high (true) or active low (false)
	LON2Cfg	<LON2Cfg>**1**</LON2Cfg>	U16	C	Sets the mode of operation of LASERON2 0 - LASERON2 == !LASERON1 1 - LASERON2 == LASERON1 & !LasersEnabled 2 - LASERON2 == !LasersEnabled 3 - LASERON2 == Asserted all of the time Lasers are enabled or disabled via the streaming job command <set id=”EnableLaser”>{false,true}</set>
	LMOD1High	<LMOD1High>**false**</LMOD1High>	BOOL	C	LASERMOD1 signal is active high (true) or active low (false)
	LMOD2High	<LMOD2High>**false**</LMOD2High>	BOOL	C	LASERMOD2 signal is active high (true) or active low (false)
	LFPKHigh	<LFPKHigh>**false**</LFPKHigh>	BOOL	C	LASERFPK signals is active high (true) or active low (false)
	LsrEnaDly	<LsrEnaDly>**4000**</LsrEnaDly>	U16	C	The time require for enabling the laser prior to actual use (usec). Set the time that the signal LASERENABLE is asserted prior to a marking operation.
	LsrEnaTmo	<LsrEnaTmo>**4000**</LsrEnaTmo>	U16	C	The time that the signal LASERENABLE will remain asserted after a marking operation. If a subsequent marking operation is started prior to the expiration of this time, then LASERENABLE will remain asserted and the marking operation will begin immediately without the cost of another LsrEnaDel.
	LsrModDly	<LsrModDly>**20**</LsrModDly>	U16	C	The time from the assertion of LASERON to the emission of laser pulses.
	LsrTiming	<LsrTiming>**50**</LsrTiming>	U16	C	The number of 20ns intervals that make up a laser timing “tick”
	LsrPwrDly	<LsrPwrDly>**1700**</LsrPwrDly>	U16	C	The time required after laser power is changed until the laser power has settled (usec). Used when constructing jobs that manipulate the laser power.
	LsrPwrMode	<LsrpwrMode>**8bit**</LsrPwrMode>	STR	C	Sets the mode of the digital power level port 8bit - The laser power setting is driven as an 8-bit word 7bit - The laser power setting is driven as a 7-bit word. The least significant bit is redefined as a data strobe going high for 100usec and then low after a data word settling time of 100us
	FpsPos	<FpsPos>**30**</FpsPos>	U16	C	The time between the assertion of the LASERFPK signal and the generation of LASERMOD pulses (µsec)
	FpsWidth	<FpsWidth>**0**</FpsWidth>	U16	C	The width of the LASERFPK pulse (µsec)
	TickleWidth1	<TickleWidth1>**5**</TickleWidth1>	U16	C	The width of the LASERMOD1 pulses during standby (µsec)
	TickleFreq1	<TickleFreq1>**1**</TickleFreq1>	U16	C	The frequency of the LASERMOD1 pulses during standby (KHz)
	TickleWidth2	<TickleWidth2>**5**</TickleWidth2>	U16	C	The width of the LASERMOD2 pulses during standby (µsec)

Purpose	The laser configuration defines the properties of the laser in use with the EC device				
	XML Tag	XML Example Text	Type	Use	Description
Definition	TickleFreq2	<TickleFreq2>**1**</TickleFreq2>	U16	C	(Future) The frequency of the LASERMOD2 pulses during standby (KHz). Currently TickleFreq1 and TickleFreq2 must be set equal to each other
	InitBits	<InitBits>**1**</InitBits>	U16	C	Initialization value for lasers with digital power control
	InitLaser	<InitLaser>**false**</InitLaser>	BOOL	C	Laser requires explicit initialization via serial comm (true)
	InitType	<InitType>**0**</InitType>	U16	C	Laser communications type: 0 = RS-232 Serial, 1 = Ethernet
	BaudRate	<BaudRate>**0**</BaudRate>	U32	C	Baud rate of the serial interface on the laser
	InitStrDelim	<InitStrDelim>**”,”**</InitStrDelim>	CHR	C	Delimiter character separating command and argument tokens in the InitString
	InitStrEOL	<InitStrEOL>**”\n”**</InitStrEOL>	CHR	C	Line termination character used by the laser command interpreter
	InitStrings	<InitStrings> <InitString>**ab**</InitString> <InitString>**cd**</InitString> <InitString>**ef**</InitString> </InitStrings>	STR	C	A list of initialization strings to be sent to the laser. The list may be arbitrarilly long
	DeinitStrings	<DeinitStrings> <DeinitString>**zy**</DeinitString> <DeinitString>**xw**</DeinitString> <DeinitString>**vu**</DeinitString> </DeinitStrings>	STR	C	A list of deinitialization strings to be sent to the laser. The list may be arbitrarilly long
	CorrTable	<CorrTable> <Entry>**0**</Entry> <Entry>**1**</Entry> . . . <Entry>**255**</Entry> </CorrTable>	I8	C	*A list of laser power linearization values. Laser power has a logical range of 0-255 and as a power change is requested, the logical power value is used to index this table and the selected entry is used as the actual “corrected’ value. (Future)*
		</Data>			End LaserConfigData
Explanation	These values are normally set by the integrator and not intended to be altered by a marking application.				
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()				

Lens Configuration

Purpose	The lens configuration describes the properties of the lens in use with the EC device.				
	XML Tag	XML Example Text	Type	Use	Description
Definition	Data	<Data type="LensConfigData" rev=”1.0”>			LensConfigData identifier
	LensName	<LensName>**S4LFT0163**</LensName>	STR	A	Used by the head integrator to identify a particular lens model
	CalFlag	<CalFlag>**false**</CalFlag>	BOOL	A	Used by an application to indicate that this lens can be calibrated.
	KFactor	<KFactor>**500**</KFactor>	U32	A	Scale factor relating the X and Y galvo command signals to the distance traveled by the laser (bits/mm)
	ZMode	<ZMode>**0**</ZMode>	U16	A	Specifies the Z axis operational mode:

Name	Value	Description
2D	0	No Z axis is present in the system and only X and Y vector data is used.
3D	1	Z axis is present and the Z position is the Z axis job data adjusted by the interpolated value from the Z axis component of the currently active correction table. The Z axis moves smoothly to the target position over the same time period it takes to move to the X-Y target position.

	XML Tag	XML Example Text	Type	Use	Description
	ZKFactor	<ZKFactor>**500**</ZKFactor>	U32	A	Scale factor relating the Z (focus) actuator command signals to the focal plane displacement (bits/mm)
	FocalLen	<FocalLen>**163**</FocalLen>	U32	A	Focal length of the lens (mm)
	Aperture	<Aperture>**15**</Aperture>	U32	A	Laser beam diameter entering the lens (mm)
	Tbl1XOff	<Tbl1XOff>**0**</Tbl1XOff>	I16	C	X axis offset to be applied to correction table 1 (bits)
	Tbl1YOff	<Tbl1YOff>**0**</Tbl1YOff>	I16	C	Y axis offset to be applied to correction table 1 (bits)
	Tbl1XGain	<Tbl1XGain>**1.0**</Tbl1XGain>	FLT	C	X axis gain to be applied to correction table 1
	Tbl1YGain	<Tbl1YGain>**1.0**</Tbl1YGain>	FLT	C	Y axis gain to be applied to correction table 1
	Tbl1Rotation	<Tbl1Rotation>**0.0**</Tbl1Rotation>	FLT	C	Field rotation to be applied to correction table 1 (degrees)
	Tbl2XOff	<Tbl2XOff>**0**</Tbl2XOff>	I16	C	X axis offset to be applied to correction table 2 (bits)
	Tbl2YOff	<Tbl2YOff>**0**</Tbl2YOff>	I16	C	Y axis offset to be applied to correction table 2 (bits)
	Tbl2XGain	<Tbl2XGain>**1.0**</Tbl2XGain>	FLT	C	X axis gain to be applied to correction table 2
	Tbl2YGain	<Tbl2YGain>**1.0**</Tbl2YGain>	FLT	C	Y axis gain to be applied to correction table 2
	Tbl2Rotation	<Tbl2Rotation>**0.0**</Tbl2Rotation>	FLT	C	Field rotation to be applied to correction table 2
		</Data>			End LensConfigData

Purpose	The lens configuration describes the properties of the lens in use with the EC device.				
	XML Tag	XML Example Text	Type	Use	Description
Explanation	These values are normally set by the integrator and not intended to be altered by a marking application. Note that the Tbl{1,2} offset, gain and rotation factors are intended to be used by the integrator to correct for system alignment issues and for the effects of the different wavelengths of light used for marking (table 1) and pointing (table 2). User level adjustments to the imaging field are performed through the use of the UserConfigData. The order of application of the factors is as follows: $\begin{bmatrix}X\\Y\end{bmatrix} = \begin{bmatrix}XGain\cdot\cos(Rotation) & XGain\cdot(-\sin(Rotation))\\ YGain\cdot\sin(Rotation) & YGain\cdot\cos(Rotation)\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}Xoff\\Yoff\end{bmatrix}$				
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()				

Correction Tables

Purpose	The correction table contains values to adjust laser location based on the lens distortion and laser galvo configuration			
Definition	XML Tag	XML Example Text	Type	Description
	Data	<Data type="CorrTableData" rev="1.0">		CorrTableData dentifier
	x-axis	<x-axis>**203,195,161,…,-174,-190,-201**</x-axis>	I32	X-Axis Correction Data 4225 values defining the x-axis correction starting in the lowest negative coordinate (lower left Cartesian quadrant) to the highest positive coordinate (upper right Cartesian quadrant).
	y-axis	<y-axis>**337,323,288,…,-288,-323,-337**</y-axis>	I32	Y-Axis Correction Data 4225 values defining the Y-axis correction starting in the lowest negative coordinate (lower left Cartesian quadrant) to the highest positive coordinate (upper right Cartesian quadrant).
	z-axis	<z-axis>**2,2,1,…,4,5,5**</z-axis>	I32	Z-Axis Correction Data 4225 values defining the Z-axis correction starting in the lowest negative coordinate (lower left Cartesian quadrant) to the highest positive coordinate (upper right Cartesian quadrant).
		</Data>		End CorrTableData
Explanation	Correction table data may be changed by an application, but is normally not. This data is usually provided by a marking head integrator using the characteristics of the lens and laser galvo configuration. Correction table data may also be sent to the EC1000 using the sendStreamData() method. In this case, however, the data is not persistent and will be lost after session logout or reboot.			
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()			

User Configuration

Purpose	The User Confguration table contains values that are completely under the control of a marking application			
Definition	XML Tag	XML Example Text	Type	Description
	Data	<Data type="UserConfigData" rev="1.0">		UserConfigData identifier
	XOff	<XOff>**0**</XOff>	I16	Offset to be applied to all X-Axis coordinates (bits)
	YOff	<YOff>**0**</YOff>	I16	Offset to be applied to all Y-Axis coordinates (bits)
	XGain	<XGain>**1.0**</XGain>	FLT	Gain factor to be applied to all X-axis coordinates
	YGain	<YGain>**1.0**</YGain>	FLT	Gain factor to be applied to all Y-axis coordinates
	Rotation	<Rotation>**90.0**</Rotation>	FLT	Rotation transformation to be applied to the X-Y field
	UserVar1	<UserVar1>**ABC**</UserVar1>	ANY	General purpose user variable
	UserVar2	<UserVar2>**123**</UserVar2>	ANY	General purpose user variable
	UserVar3	<UserVar3>**4.56**</UserVar3>	ANY	General purpose user variable
	UserVar4	<UserVar4>**true**</UserVar4>	ANY	General purpose user variable
	UserVar5	<UserVar5>**false**</UserVar5>	ANY	General purpose user variable
	UserVar6	<UserVar6>**"text"**</UserVar6>	ANY	General purpose user variable
		</Data>		End UserConfigData
Explanation	This data is intended to be used by a marking application as needed. The offset, gain and rotation variables are independent of and additive to the equivalent lens correction table adjustment factor defined in the LensConfigData able. The general purpose user variables can be used to store any information that a marking application wishes to make persistant across reboots of the controller. It is up to the application to interpret the UserVar data as required.			
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()			

Performance Adjustments

Purpose	The Performance Adjustments table contains values that are used to adjust job parameters while the job is executing			
Definition	XML Tag	XML Example Text	Type	Description
	Data	<Data type="PerformanceMatrixData" rev=”1.0”>		PerformanceMatrixData identifier
	LaserPower	<LaserPower>**1.0**</LaserPower>	FLT	Scale factor to be applied to the laser power value specified in the job
	LaserPowerComp	<LaserPowerComp>**1.0**</LaserPowerComp>	FLT	Secondary scale factor to be applied to the laser power value specified in the job
	PulseWidth	<PulseWidth>**1.0**</PulseWidth>	FLT	Scale factor to be applied to the laser pulse width specified in the job
	Period	<Period>**1.0**</Period>	FLT	Scale factor to be applied to the laser pulse period specified in the job
	MarkSpeed	<MarkSpeed>**1.0**</MarkSpeed>	FLT	Scale factor to be applied to the MarkSpeed specified in the job
	XOffset	<XOffset>**0**</XOffset>	I16	Offset to be applied to all X coordinates (bits)
	YOffset	<YOffset>**0**</YOffset>	I16	Offset to be applied to all Y coordinates (bits)
	ZOffset	<ZOffset>**0**</ZOffset>	I16	Offset to be applied to all Z coordinates (bits)
		</Data>		End PerformanceMatrixData
Explanation	This data is intended to be used by a marking application as needed. These factors are applied to the job parameters at run-time. They are typically used to adjust marking performance without requiring alterations to the jobs. This is of particular value when jobs are stored locally and adjustments need to be made to compensate for laser degradation on a particular machine.			
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()			

6.3.4 Send Fixed Data

Data that was retreived via Session.requestFixedData can be modified and sent back to the EC1000 for storage.

Command	ILecSession.sendFixedData
Purpose	Send fixed data to an EC device session

Usage	ILecSession.sendFixedData(
	ByVal pstrData As String, // The data sent to the EC device. The string supplied contains an XML
	// representation of the data.
	ByVal pstrStorageName As String // File *name* of the data file. File *path* is constructed by the API as follows:
	// <PermStoragePath>\LEC\Config\<pstrStorageName>.xml
	// where <PermStoragePath> is defined in the SysInfoData for the selected
	// EC1000 and <pstrStorageName> is the name of the selected fixed data file
	// as stored on the EC1000 without the ".xml" extension.
	ByVal puiTimeout As Unsigned Long // Duration for attempting call in seconds.
	) As Unsigned Long

Explanation	Data retrieved via the Session.requestFixedData() method may be modified and passed back to the controller for local storage. That data will then be used the next time the module is rebooted. Note that not all data is changeable by the application since some of it is set at the factory. See the Session Fixed Data Definitions section for details. An application should wait on event "FixedDataDone" to be assured the the updated data has been processed by the EC1000 and is ready for subsequent actions.
Returns	0 – Success 10 – Network connection not established 12 – Location for data not found on EC device 13 – Cannot access the remote location where the data is written on the EC device 14 – Cannot access the local source location where the data resides 15 – The data type transmitted is unknown 21 - Licensing is not available 22 - Licensing is denying access to the Broadcast feature
See also	ILecSession.requestFixedData

6.3.5 Send Streaming Job Data

The primary inteface for interacting with the controller is the Session.sendStreamData() method. This method streams data to the controller as fast as the network and buffering systems allow. Buffering is distributed between the host operating system, the EC1000 operating system, the EC1000 control software, and finally, the marking engine input FIFO. The method returns as soon as the data is passed to the downstream communications system and has been transferred to the target EC1000. Once this method returns, subsequent calls can be made to keep the data ”pipeline” full with marking data. This technique ensures continuous marking operation without pauses.

Command	ILecSession.sendStreamData
Purpose	Send streaming data to an EC device session.
Usage	Session.sendStreamData(ByVal pstrData as String, // The data sent to the EC device. The string supplied contains an XML // representation of the data. ByVal puiTimeout As Unsigned Long // Duration for attempting call in seconds) As Unsigned Long
Explanation	Marking jobs are specified as sequences of data that represent instructions to the controller to set operational parameters, activate the laser steering galvos in both marking and non-marking modes, to interact with external devices, and to send event information back to a listening application. The job data is specified in an XML string, which is defined in the Streaming Job Data Defintion section. Job execution by the controller starts as soon as the job data is received by the module and continues for as long as job data is available. Very large jobs can be partitioned into logical chunks, such as at marking object boundaries, and streamed out to the device as buffering on the host and target allow. Since the execution of the job and the process of streaming the data of the job are asynchronous and overlapped, it is possible to maintain continuous job execution with no pauses.
Returns	0 – Success 1 – Access writing the stream data is denied 2 – Could not establish a connection 3 – No connection 15 – Unknown data. Error in data format 21 - Licensing is not available 23 - Invalid command or parameter in the job stream data
See also	

Streaming Job Data Defintion

Job data contains both actionable commands that direct the marking engine to perform specific operations, and parametric data that affects how the EC1000 hardware behaves. To minimize the number of XML identifier tags to express a job, the XML definition make use of two types of constructs. All actionable commands use specific XML tag names to identify the action, followed by a comma separate list of argument values. The *Set* tag is used with an identifier for a parameter followed by a comma separated list of values. For example, the following XML text expresses a simple job that draws a square box.

XML Text	Description
<Data type=”JobData” rev=”1.0”>	Job data type declaration
<Set id=”JumpSpeed”>**15,30**</Set>	The parameter ”JumpSpeed” is set to 30 bits per each 15µ sec update period
<Set id=”MarkSpeed”>**15,10**</Set>	The parameter ”MarkSpeed” is set to 10 bits per each 15µ update period
<JumpAbs>**-5000,-5000**</JumpAbs>	Move laser galvos to the absolute position -5000, -5000 with the laser off
<MarkAbs>**-5000,5000**</MarkAbs>	Move laser galvos to the absolute position -5000, 5000 with the laser on
<MarkAbs>**5000,5000**</MarkAbs>	Move laser galvos to the absolute position 5000, 5000 with the laser on
<MarkAbs>**5000,-5000**</MarkAbs>	Move laser galvos to the absolute position 5000, -5000 with the laser on
<MarkAbs>**-5000,-5000**</MarkAbs>	Move laser galvos to the absolute position -5000, -5000 with the laser on
</Data>	End job data

Job Command Tags

Command Tag	Description/Example XML Syntax	Parameters	Units	Min	Max
BeginJob	Generates a BeginJob application event (see section 6.3.12) when executed by the marking engine. The <JobDataCntr> parameter in the StatInfoData broadcast packet is reinitialized to zero. BeginJob automatically sets the system BUSY signal. Example: <BeginJob></BeginJob>	N/A			
EndJob	Generates an EndJob application event (see section 6.3.12) when executed by the marking engine. The system BUSY signal is automatically cleared. Example: <EndJob></EndJob>	N/A			

Command Tag	Description/Example XML Syntax	Parameters	Units	Min	Max
JumpAbs	Move laser galvos to the absolute position with the laser off Example: <JumpAbs>**-5000,-5000**</JumpAbs>	X Coordinate	bits	-32768	32767
		Y Coordinate	bits	-32768	32767
		Z Coordinate (optional) If Z is absent, then the Z value is assumed to be zero	bits	-32768	32767
MarkAbs	Move laser galvos to the absolute position with the laser on Example: <MarkAbs>**-5000,5000,200**</MarkAbs>	X Coordinate	bits	-32768	32767
		Y Coordinate	bits	-32768	32767
		Z Coordinate (optional) If Z is absent, then the Z value is assumed to be zero	bits	-32768	32767
LaserOn	Turn the laser on for the duration provided. Example: <LaserOn>**1000**</LaserOn>	Duration – length of time to turn the laser on	µsec	1	$(2^{32}-1)$ /50
WriteAnalog	Commands the analog output port to a new value. Port 0 is the Laser Power port (A1), and Port 1 is the auxilliary analog output port (A2). A write to port 0 will incur the LaserPowerDelay (see next section) Example: <WriteAnalog>**1,344**</WriteAnalog>	portNumber – analog output port identifier	N/A	0	1
		value – new port value	bits	0	4095
WriteDigital	Commands the digital output port to a new value. Port 0 is the reserved SPAREOUT port. Ports 1-4 select the signals USEROUT1-4 respectively. Port 5 is the MRKINPRG signal. Example: <WriteDigital>**3,1**</WriteDigital>	portNumber – port identifier	N/A	0	5
		value – 0 (off) and 1 (on)	bool	0	1
LaserSignalOn	Turns laser on immediately Example: <LaserSignalOn></LaserSignalOn>	N/A			
LaserSignalOff	Turns laser off immediately Example: <LaserSignalOff></LaserSignalOff>	N/A			
JobMarker	Puts the data value into the broadcast status data <JobMarker> element. Typically used to track job execution progress. Example: <JobMarker>**35**</JobMarker>	value - application defined marker value	N/A	0	65536
ApplicationEvent	Application specific command defining an event. Events are used to notify the application that a certain point in the execution of the job has been reached. Events are handled by the application using the Session.OnMessageEvent method (see section 6.3.12). Example: <ApplicationEvent>**100,345**</ApplicationEvent>	param1 – First application-specific parameter	N/A	0	65536
		param2 – Second application-specific parameter	N/A	0	$2^{32}-1$
WaitForIO	Wait for the digtal port value to be set. Job execution will pause until the external signal is in the state or changes to the state as specified. If a timeout occurs, an exception event is generated and the WaitForIOTimeout message event will be passed back to the application. Example: <WaitForIO>**2,1,100000,5000**</WaitForIO>	portNumber – port identifier. Port 0 selects the STRTMRK input signal. Ports 1-4 select the USERIN1-USERIN4 signals	N/A	0	4
		levelPolarity – 0 LowLevel, 1 HighLevel, 2 RisingEdge, 3 FallingEdge	N/A	0	3
		timeout – abort wait if time exceeds the value. If timeout is zero, then wait indefinitely.	µsec	1	$(2^{32}-1)$ /50
		debounce – debounce the external signal for this period of time	msec	1	65535
LongDelay	Delay all operations for the duration provided Example: <LongDelay>**10000**</LongDelay>	Duration – length of time to sleep	µsec	1	$(2^{32}-1)$ /50
MotfEnable	Enables or disables Mark-on-the-fly (Motf) tracking. Upon enabling, the Motf encoder counter is zeroed and then starts incrementing or decrementing as motion is detected by the encoder. If in simulate mode (see MotfMode), the counter is incremented at a 1Mhz rate. Example: <MotfEnable>1</MotfEnable>	state - 0 disabled, 1 - enabled	N/A	0	1
MotfWaitForCount	The Motf counter is cleared, and then the job pauses until the absolute value of the Motf encoder counter reaches the specified value. The counter is automatically cleared when the value is reached. Example: <MotfWaitForCount>**24557**</MotfWaitForCount>	count - encoder count	cts	-2^{31}	$2^{31}-1$

Command Tag	Description/Example XML Syntax	Parameters	Units	Min	Max
MotfResetJump	When encountered, Motf is disabled, a Jump is performed to the specified coordinates, the Motf counter is cleared, and then Motf is re-enabled. Example: <MotfResetJump>**-23000,400,0,200**</MotfResetJump>	X, Y, Z - coordinate	bits	-32768	32767
		JumpDelay	usec	0	65535
Draw	Specifies the start of a "draw" list. Up to 32 vertices of a polygon may be specified to be marked in a loop that will run continuously until any other instruction or an Abort message is received by the EC1000. Example: <Draw> <Vertex>1000,1000</Vertex> <Vertex>1000,-1000</Vertex> <Vertex>-1000,-1000</Vertex> <Vertex>-1000,1000</Vertex> </Draw>	X, Y coordinate	bits	-32768	32767
WobbleEnable	Enables or disables the wobble function. Wobble parameters should have already been set using the <Setid="Wobble"> parameter Example: <WobbleEnable>1</WobbleEnable>	state - 0 disabled, 1 - enabled	N/A	0	1
Set	Example: <Set id="MarkSpeed">**500**</set>	Sets a job parameter (see next section)			

Job Parameter Identifiers

The following list of identifiers is valid for use with the *Set* tag. Multiple values for parameters are expressed in a comma separated list:.

Param Identifier	Description	Arguments	Units	Min	Max
JumpDelay	Set the delay for moving the laser Example: <set id="JumpDelay">**150**</set>	duration – length of time to delay	µsec	0	
JumpSpeed	Set the jump speed of the laser. The parameters are normally derived from an application speed setting expressed as bits/msec or some other appropriate ratio. Example: <set id="JumpSpeed">**15,30**</set>	stepPeriod – the duration between each micro-step, i.e. how often the galvo position command is updated	µsec	15	65535
		stepSize – the length traveled for each micro-step	bits	1	65535
LaserOffDelay	Set the delay for turning off the laser when marking Example: <set id="LaserOffDelay">**100**</set>	duration – length of time to delay	laser timing ticks	0	65535
LaserOnDelay	Set the delay for turning on the laser when marking relative to micro-vector generation. A negative value means that LASERON is asserted before micro-vectoring begins. Example: <set id="LaserOnDelay">**200**</set>	duration – length of time to delay relative to the start of micro-vectoring.	laser timing ticks	-32768	32767
LaserPower	Set the level of the digital laser power port Example: <set id="LaserPower">**200**</set>	powerValue - value to set the digital laser power port. If the value is different from the last LaserPower command, then the LaserPowerDelay delay is invoked.	counts	0	255
LaserPowerDelay	Delay after changing power setting. A default value can be set in the LaserConfig fixed data as parameterLsrPwrDly. Example: <set id="LaserPowerDelay">**125**</set>	duration - length of time to delay after setting LaserPower or executing WriteAnalog for port 0	µsec	0	$(2^{32}$-1) /50
MarkDelay	Set the delay for ending a series of marks Example: <set id="MarkDelay">**150**</set>	duration - length of time to delay	µsec	0	65535
MarkSpeed	Set the marking speed of the laser Example: <set id="MarkSpeed">**15,70**</set>	stepPeriod – the duration between each micro-step	µsec	15	65535
		stepSize – the length traveled for each micro-step	bits	1	65535
PolyDelay	Set the delay between two marks. Example: <set id="PolyDelay">**150**</set>	duration – length of time to delay	µsec	0	65535

Param Identifier	Description	Arguments	Units	Min	Max
LaserTiming	Define the value of a laser timing “tick” unit. All laser timing values are in units of LaserTiming ticks. Typically, the laser timing tick is set to 1usec so that other laser timing parameters can be more easily interpreted. Example: <set id=”LaserTiming”>**50**</set>	value – number of 20ns clock period increments in a laser timing “tick”	20ns increments	5	500
LaserPulse	Set the laser ON pulse settings of the laser Example: <set id=”LaserPulse”>**1,50,100**</set>	value – laser modulation signal identification: 1 for LASERMOD1 2 for LASERMOD2	N/A	1	2
		width – when the laser is ON, the width of the laser modulation pulse	laser timing ticks	0	65535
		period – when the laser is ON, the period of the laser modulation pulse train for both LASERMOD1 and LASERMOD2.	laser timing ticks	0	65535
VariPolyDelayFlag	Set if using variable polygon delay values. If variable polygon delays are used, then the PolyDelay value is adjusted proportional to the angular change in the next segment of the poly-vector. Example: <set id=”VariPolyDelayFlag”>**true**</set>	value - variable polygon delay enabled state: false (disabled) true (enabled)	Bool	0	1
VariJumpDelay	Below a given jumpLengthLimit, the jump delay is linearly scaled down from the JumpDelay value to the minimumDelay as the jump distance approachs zero. Example: <set id=”VariJumpDelay”>**50,2000**</set>	minimumDelay – minimum length of time for a jump delay	laser timing ticks	0	65535
		jumpLengthLimit – jump length threshold below which the variable jump delay will be applied	bits	1	65535
Wobble	Allows varying line width during *Mark* operations. Example: <set id=”Wobble”>**250,10**</set>	period – period of the wobble movement	µsec	20	65535
		amplitude – amplitude of the circular wobble movement	bits	1	32767
MotfCalFactor	Relates laser positioning bits to motion encoder counts. A default value for MotfCalFactor can be set in the ControllerData fixed data structure. Example: <set id=”MotfCalFactor”>**23.345**</set>	value - calibration factor. A negative number corresponds to a downward counting encoder tracking forward motion.	bits/ count	-32768.0	32767.0
MotfMode	Defines how Motf position information is derived. If Encoder is selected, the quadrature encoder inputs are used. If Simulate is selected, a 1Mhz clock is used to increment the encoder counter. A default value for MotfCalFactor can be set in the ControllerData fixed data structure. Example: <set id=”MotfMode”>**0**</set>	mode - position tracking mode 0 - Use encoder 1 - Simulate endoder	N/A	0	1
MotfDirection	MOTF orientation and direction in degrees. A default value for MotfDirection can be set in the ControllerData fixed data structure. Example: <set id=”MotfDirection”>**270**</set>	direction - target travel direction relative to a cartesian coordinate system: 0 - left to right in the X axis 90 - bottom to top in the Y axis 180 - right to left in the X axis 270 - Top to bottom in the Y axis	deg	0	270
ActiveCorrectionTable	Set the active current correction table. Example: <set id=”ActiveCorrectionTable”>**1**</set>	table - correction table selector (1-4) Only tables 1 and 2 should be selected during job execution. Tables 3 and 4 are to used only for loading alternate data for the XY2-100 interface when tables 1 & 2 are selected, respectively	N/A	1	4

<table>
<tr><th>Param Identifier</th><th>Description</th><th>Arguments</th><th>Units</th><th>Min</th><th>Max</th></tr>
<tr><td rowspan="2">LaserModeConfig</td><td>Set the laser configuration bitmask.
Example:
  <set id="LaserModeConfig">**0x1FF**</set></td><td colspan="4">bitmask - Laser configuration settings, bit definitions below
Default values for the the individual signals can be set by setting the LaserConfigData elements: LENAHigh, LONHigh, LMOD1High, LMOD2High, LFPKHigh</td></tr>
<tr><td colspan="5"><table>
<tr><th>Name</th><th>Value</th><th>Definition</th></tr>
<tr><td>LASERON1 polarity</td><td>0x0001</td><td>0=active high, 1=active low</td></tr>
<tr><td>LASERON2 polarity</td><td>0x0002</td><td>0=active high, 1=active low</td></tr>
<tr><td>LASERMOD1 polarity</td><td>0x0008</td><td>0=active high, 1=active low</td></tr>
<tr><td>LASERMOD2 polarity</td><td>0x0010</td><td>0=active high, 1=active low</td></tr>
<tr><td>LASERFPK polarity</td><td>0x0020</td><td>0=active high, 1=active low</td></tr>
<tr><td>LASERENABLE polarity</td><td>0x0040</td><td>0=active high, 1=active low</td></tr>
<tr><td>LSRPWRDOUT polarity</td><td>0x0080</td><td>0=active high, 1=active low</td></tr>
<tr><td>Laser activate</td><td>0x0100</td><td>1=activate (enable) laser output signals</td></tr>
</table></td></tr>
<tr><td rowspan="2">LaserFPK</td><td rowspan="2">Set the LASERFPK signal timing. Default values for these settings can be set in the LaserConfig fixed data as the element names: FPSPos and FPSWidth.
Example:
  <set id="LaserFPK">**-100,10**</set></td><td>position – the delay from the leading edge of LASERON to the assertion of the LASERFPK signal</td><td>laser timing ticks</td><td>-32768</td><td>32767</td></tr>
<tr><td>length – duration of assertion of the LASERFPK signal</td><td>laser timing ticks</td><td>0</td><td>65535</td></tr>
<tr><td>JobDataCntr</td><td>Set the job data counter to the specified value. The job data counter is incremented as each 32-bit data element of the job stream is processed by the marking engine. This is useful for tracking how much data the marking engine has processed at any given time. The current value of the counter is reported in the System Status broadcast data as element name JobDataCntr.
Example:
  <set id="JobDataCntr">**0**</set></td><td>value – counter value (only accepts zero for now)</td><td>N/A</td><td>0</td><td>0</td></tr>
<tr><td rowspan="3">LaserStandby</td><td rowspan="3">Set the standby settings of the laser. Default values for these settings can be set in the LaserConfig fixed data as the element names: TickleWidth1, TickleFreq1, TickleWidth2, TickleFreq2
Example:
  <set id="LaserStandby">**2,10,100**</set></td><td>value – laser modulation signal identification
  1 for LASERMOD1
  2 for LASERMOD2</td><td>N/A</td><td>1</td><td>2</td></tr>
<tr><td>width – when the laser is ON, the width of the laser modulation pulse</td><td>laser timing ticks</td><td>0</td><td>65535</td></tr>
<tr><td>period – when the laser is ON, the period of the laser modulation pulse train</td><td>laser timing ticks</td><td>0</td><td>65535</td></tr>
<tr><td>LaserModDelay</td><td>Set the modulation delay of the laser.
Example:
  <set id="LaserModDelay">**25**</set></td><td>delay – the delay from the leading edge of LASERON to the output of the first pulse on the LASERMOD1 signal</td><td>laser timing ticks</td><td>0</td><td>65535</td></tr>
<tr><td>EnableLaser</td><td>Set the laser active state
Example:
  <set id="EnableLaser">**true**</set></td><td>active – laser active state
  false - disabled
  true - enabled</td><td>Bool</td><td>false</td><td>true</td></tr>
<tr><td>LaserEnableDelay</td><td>Set the time required to enable the laser prior to marking. A default value can be set in the LaserConfig fixed data as parameter LsrEnaDly.
Example:
  <set id="LaserEnableDelay">**4000**</set></td><td>delay – the delay from the leading edge of LASERENABLE to the leading edge of LASERON</td><td>µsec</td><td>0</td><td>65535</td></tr>
<tr><td>LaserEnableTimeout</td><td>Set the timeout for LASERENABLE to de-assert. A default value can be set in the LaserConfig fixed data as parameter LsrEnaTmo.
Example:
  <set id="LaserEnableTimeout">**4000**</set></td><td>delay – the timeout from the trailing edge of LASERON to when LASERENABLE is de-asserted</td><td>µsec</td><td>0</td><td>65535</td></tr>
<tr><td>GalvoDACConfig</td><td>Set the analog command output configuration for the laser galvo servo controllers using a bitmask.
Example:
  <set id="GalvoDACConfig">**0x6**</set>
      // sets the X & Y range to ±10V Z to ±5V</td><td colspan="4">bitmask – Bitmask which defines analog output configuration. The mask is defined as follows:
  Bits 1..0 encode the range of the X & Y DACs
  Bits 3..2 encode the range of the Z DAC
The range encoding is as follows:
  00 = ±2.5V, 01 = ±5V, 10 = ±10V</td></tr>
</table>

<table>
<tr><th>Param Identifier</th><th>Description</th><th>Arguments</th><th>Units</th><th>Min</th><th>Max</th></tr>
<tr><td>LaserPipelineDelay</td><td>Set the time that all laser signals are time shifted relative to the issuance of galvo position commands. This delay is useful for compensating for digital servo controllers that have an inherent processing delay time from when the command input is applied to when the mirrors actually move.
Example:
<set id=”LaserPipelineDelay”&gt;<b>1500</b>&lt;/set&gt;</td><td>delay – the delay that all laser control signals are time shifted relative to micro-vectoring operations.</td><td>laser timing ticks</td><td>0</td><td>4095</td></tr>
<tr><td>ServoConfig</td><td>Set the configuration of the X, Y and Z servo control interface.
Example:
<set id=”ServoConfig”&gt;<b>0x4</b>&lt;/set&gt;
// Enable X & Y galvos, but not Z. All polarities
// active low, no exceptions are to be generated.</td><td colspan="4">bitmask - Hexidecimal bitmask that defines the configuration of the laser galvo servo interface. See definitions, below
.
<table>
<tr><th>Name</th><th>Value</th><th>Definition</th></tr>
<tr><td>X_SERVO_EN and Y_SERVO_EN polarity</td><td>0x0001</td><td>0=active high, 1=active low</td></tr>
<tr><td>Z_SERVO_EN polarity</td><td>0x0002</td><td>0=active high, 1=active low</td></tr>
<tr><td>Enable X, Y Servos</td><td>0x0004</td><td>1=enable servos, 0=disable</td></tr>
<tr><td>Enable Z Servo</td><td>0x0008</td><td>1=enable servos, 0=disable</td></tr>
<tr><td>X_SERVO_RDY and Y_SERVO_RDY polarity</td><td>0x0010</td><td>0=active high, 1=active low</td></tr>
<tr><td>Z_SERVO_RDY polarity</td><td>0x0020</td><td>0=active high, 1=active low</td></tr>
<tr><td>X, Y Not-ready exception enable</td><td>0x0040</td><td>1=enable exception event generation is X or Y servo becomes not ready</td></tr>
<tr><td>Z Not-ready exception enable</td><td>0x0080</td><td>1=enable exception event generation if Z servo becomes not ready</td></tr>
</table></td></tr>
</table>

6.3.6 Save Job Data

When a job has been constructed and tested using on-line worktation facilities, it can be sent to the EC1000 for storage so that it can be run when the controller is placed in “local” mode.

Command	ILecSession.saveJobData
Purpose	Send job data for storage in the EC1000 Flash memory or USB device
Usage	Session.saveJobData(ByVal uiTargetLoc as Unsigned Long, // Storage location: 1 == Flash on EC1000, 2 == USB Flash device on EC1000 ByVal pstrStorageName as String, // Name to use as the file name ByVal pstrJobData As String, // XML representation of the job data ByVal uiAccessType As Unsigned Long, // Access type: 0 == Overwrite, 1 == Append *(future)* ByVal uiTimeout As Unsigned Long // Duration for attempting call in seconds) As Unsigned Long
Explanation	Job data is compiled and stored on the target EC1000 Flash file system for use in Local Mode
Returns	0 – Success
See also	LecSession.manageJobData(), ILecSession.requestJobNameList()

6.3.7 Manage Job Data

A job has been stored on the EC1000 can be renamed or deleted using this command.

Command	ILecSession.manageJobData
Purpose	Manages jobs that have been stored on the EC1000
Usage	Session.manageJobData(ByVal uiTargetLoc as Unsigned Long, // Storage location: 1 == Flash on EC1000, 2 == USB Flash device on EC1000 ByVal pstrCurStorageName as String, // Current file name ByVal pstrNewStorageName as String, // New file name ByVal uiActionType As Unsigned Long, // Action type: 0 == Delete, 1 == Rename ByVal uiTimeout As Unsigned Long // Duration for attempting call in seconds) As Unsigned Long
Explanation	Jobs already stored on the EC1000 can be renamed or deleted.
Returns	0 – Success
See also	ILecSession.saveJobData(), ILecSession.requestJobNameList()

6.3.8 Request Job Name List

Returns a list of jobs that have been stored on the EC1000.

Command	ILecSession.requestJobNameList
Purpose	Gets a list of job names stored on the EC1000 Flash or USB Flash
Usage	Session.requestJobNameList(ByVal uiTargetLoc as Unsigned Long, // Storage location: 1 == Flash on EC1000, 2 == USB Flash device on EC1000 ByRef puiJobCount as Unsigned Long, // Number of jobs found on the target device ByRef pstrNameList as String, // XML list of jobs names ByVal uiTimeout As Unsigned Long // Duration for attempting call in seconds) As Unsigned Long
Explanation	Returns a list of jobs stored in the specified storage location on the EC1000 An example of the syntax of the list is as follows (for the EC1000 Flash device): <LECFlashJobList> <LEC jobname="JobData.wlb" /> <LEC jobname="LocalJob.wlb" /> </LECFlashJobList If the device is specified to be the USB Flash device, then <LECFlashJobList> would be <LECUSBJobList>
Returns	0 – Success
See also	ILecSession.saveJobData(), ILecSession.manageJobData()

6.3.9 Send Priority Data

Occasionally it may be neccessary to send urgent commands to the controller, such as an abort, that must bypass the data stream that is full of job data. Session.sendPriorityData provides this mechanism.

Command	ILecSession.sendPriorityData
Purpose	Send priority data to an EC device session.
Usage	Session.sendPriorityData(ByVal pstrData as String, // The data sent to the EC device. // The string supplied contains an XML representation of the data. ByVal puiTimeout As Unsigned Long // Duration for attempting call in seconds.)As Unsigned Long
Explanation	An independent and parallel communication channel is provided to the controller to pass "out-of-band" commands. This channel of communication is used to send urgent commands to the controller, such as an abort message, or pause/resume messages. The method returns as soon as the message is sent, not when the operation is actually performed on the target. Some messages, however, create response events when the action is completed, such as "Abort" and "GetRegister".
Returns	0 – Success 1 – Access writing the stream data is denied 2 – Could not establish a connection 3 – No connection 15 – Unknown data. Error in data format 21 - Licensing is not available 22 - Licensing is denying access to the Broadcast feature
See also	

Priority Data Definition

Priority messages are sent using the Session.sendPriorityData() method. The messages are expressed in XML format as described in the following table

<table>
<tr><th>Message</th><th>XML Example Syntax</th><th>Description</th></tr>
<tr><td>Abort</td><td><Data type="ServiceData" rev="1.0">
<Msg id="Abort" reason="Terminate"/>
</Data></td><td>Abort based on the reason provided. This reason causes alternative action to be taken on the EC1000 device. Abort messages result in an OnMessage event (6.3.12 Session On Message Events) being generated when the operation completes on the EC1000.
Reason:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>Job</td><td>Abort the job that is currently running</td></tr>
<tr><td>Terminate</td><td>Abort the currently running job and terminate the current session connection</td></tr>
</table></td></tr>
<tr><td>Restart</td><td><Data type="ServiceData">
<Msg id="Restart"/>
</Data></td><td>Performs a hardware reset of the EC1000. The session will be disconnected and must be re-established before additional communications is possible.</td></tr>
<tr><td>Suspend</td><td><Data type="ServiceData">
<Msg id="Suspend"/>
</Data></td><td>Suspends the execution of the job. The job is paused at the next convenient location where the lasers are off. If a <i>Mark</i> is currently in progress, it is allowed to complete ncluding poly-vector mark.</td></tr>
<tr><td>Resume</td><td><Data type="ServiceData">
<Msg id="Resume"/>
</Data></td><td>Job execution is permitted to continue.</td></tr>
<tr><td>GetRegisters</td><td><Data type="ServiceData">
<Msg id="GetRegisters"/>
</Data></td><td>Sends a request to the EC1000 to return the current values of several hardware registers on the module. Data is returned via a session OnData event (see 6.3.11 Session On Data Events) message.</td></tr>
<tr><td>SetInterlockEnable</td><td><Data type="ServiceData">
<Msg id="SetInterlockEnable" config="0x14"/>
</Data></td><td>Enables or disables the interlock function of the EC1000 based on the "config" bit pattern
Bits [3..0] represent the interlock signals INTLOCK[4..1]. A "1" enables the corresponding interlock signal.
Bit [4] is the master enable bit for the interlock function.</td></tr>
</table>

6.3.10 Session On Connect Events

The API can generate events when the API successfully "connects" to an EC1000 via the ILecSession.loginSession() method, or "disconnects" using the ILecSession.logoutSession() method. These events are accessed via the ILecSession.OnConnectEvent interface.

Command	ILecSession.OnConnectEvent
Purpose	Application and exception events returned from the EC device session.
Usage	ILecEvent Session.OnMessageEvent(ByVal pstrIPAddr As String, // The IP address of the EC1000 that the request was directed to. ByVal bState As Bool // True if connected, False if disconnected) As Unsigned Long
Explanation	

6.3.11 Session On Data Events

Priority messages that return variable data do so by generating an OnData event. In general, a request for information is made by sending a Priority Data message, e.g. "GetRegisters". When the EC1000 processes the message, it sends the requested data back through the OnData event channel.

Command	ILecSession.OnDataEvent
Purpose	Data requested from the EC1000 is returned via this interface.
Usage	ILecEvent Session.OnDataEvent(ByVal puiDataID As Unsigned Long, // Identifier of the data being returned. The identifiers are as follows; // 0 - Reserved // 1 - Registers Data // 2... Reserved ByVal puiErrorCode as Unsigned Long, // Error code returned from the EC1000. No error == 0. ByVal pstrData as String // The data sent by the EC device. // The string supplied contains an XML representation of the data.) As Unsigned Long

Explanation	Data is returned asynchronous from the request. ew
	Register Data is returned as follow:
	<Data type="HardwareState" rev="1.0">
	<ServoStatus>**0x0**</ServoStatus> // Bits 6..3 == Z, Y, X Fault. Bits 2..0 == Z, Y, X Ready.
	<XDAC>**-500**</XDAC>
	<YDAC>**-500**</YDAC>
	<ZDAC>**0**</ZDAC>
	<A1DAC>**16**</A1DAC>
	<A2DAC>**0**</A2DAC>
	<XY2Chan1>**-500**</XY2Chan1>
	<XY2Chan2>**-500**</XY2Chan2>
	<XY2Chan3>**0**</XY2Chan3>
	<XY2Status>**0x0**</XY2Status>
	<LaserControl>**0x10**</LaserControl>
	<LaserPower>**1**</LaserPower>
	<MOTFPosition>**0**</MOTFPosition>
	<DIO>**0x3FF**</DIO> // bits[3..0] == USERIN[4..1]
	// bit[5..4] == SPAREIN, STRTMRK
	// bits[9..6] == INTERLOCK[4..1]
	// bits[13..10] == USEROUT4..1]
	// bits[17..14] == SPAREOUT, LEC_ERROR, LEC_BUSY, MRKINPRG
	<DIO.IN>**0xF**</DIO.IN> // bits[3..0] == USEROUT[4..1]
	<DIO.OUT>**0x0**</DIO.OUT> // bits[3..0] == USERIN[4..1]
	<DIO.Control>**0x1**</DIO.Control> // bits[4..0] == SPAREOUT, LEC_ERROR, LEC_BUSY, MRKINPRG,
	// STRTMRK
	<DIO.Interlock>**0xF**</DIO.Interlock> // bits[3..0] == INTLOCK[4..1]
	<XVectCmd>**-500**</XVectCmd>
	<YVectCmd>**-500**</YVectCmd>
	<ZVectCmd>**0**</ZVectCmd>
	</Data>

6.3.12 Session On Message Events

Jobs can use instructions that create “events” that can be sensed by an application. Events are also generated when exception conditions occur on the EC1000, such as the response to an abort message, servo error detection, etc.

<table>
<tr><td>Command</td><td>ILecSession.OnMessageEvent</td></tr>
<tr><td>Purpose</td><td>Application and exception events returned from the EC device session.</td></tr>
<tr><td>Usage</td><td>ILecEvent Session.OnMessageEvent(
   ByVal puiPayloadHigh As Unsigned Long,   // Event data in the high order bytes.
   ByVal puiPayloadLow As Unsigned Long   // Event data in the low order bytes.
) As Unsigned Long</td></tr>
<tr><td>Explanation</td><td>
<p>Events are used to communicate asynchronous data from the controller back to the application. Events are normally produced as a result of the controller executing a Begin Job, End Job, or Application Event instruction. Exception conditions may also create a event. The data that classifies the event are passed back as two 32-bit payloads from the controller.</p>
<p>puiPayloadHigh is encoded in two 16-bit entities: puiPayloadHigh[15..0] contains the event type, and puiPayloadHigh[31..16 contains event-type specific data.</p>
<p>The following event types are supported (puiPayloadHigh[15..0]):</p>
<table>
<tr><th>Event Type</th><th>Value</th></tr>
<tr><td>Begin Job</td><td>0x0041 (65)</td></tr>
<tr><td>End Job</td><td>0x0042 (66)</td></tr>
<tr><td>Application Event</td><td>0x5040 (20544)</td></tr>
</table>
<p>If the event type is an Application Event, then puiPayloadHigh[31..16] define the application message type. The message type falls into two categories: Job messages and Exception messages. Job messages are created using the <ApplicationEvent> job command. This command takes two arguments, the first of which is a message type code, and the second of which is an arbirtary 32-bit parameter. When this command is encountered by the marking engine controller, a MessageEvent is created, and the message type code is passed back in puiPayloadHigh[31..16] and the parameter in puiPayloadLow[31..0]. The system pre-defines some <ApplicationEvent> message type codes as indicated in the following table. The Object and Task messages are intended to be used to mark boundaries in the vector list that correspond to the abstract definitions of an Object (square, circle, polygon) or a Task (complex assembly of Objects). They are not necessary for processing a job, they exist for convenience only.</p>
<p>The MarkProgress and CycleProgress events, however, are automatically generated by the EC1000 as a job is executed. The percentage complete values are calculated using information from <JobMarker> instructions that are automatically inserted into the job stream.</p>
<table>
<tr><th>Message Type</th><th>Value</th><th>Description</th><th>puiPayloadLow</th></tr>
<tr><td>BeginObject</td><td>0x0002 (2)</td><td>Beginning of a marking object</td><td>Object Identifier</td></tr>
<tr><td>EndObject</td><td>0x0003 (3)</td><td>End of a marking object</td><td>Object Identifier</td></tr>
<tr><td>BeginTask</td><td>0x0004 (4)</td><td>Beginning of a task object</td><td>Task Identifier</td></tr>
<tr><td>EndTask</td><td>0x0005 (5)</td><td>End of a task object</td><td>Task Identifier</td></tr>
<tr><td>MarkProgress</td><td>0x0006 (6)</td><td>Progress for a marking object</td><td>Percent Complete</td></tr>
<tr><td>CycleProgress</td><td>0x0007 (7)</td><td>Progress for a job</td><td>Percent Complete</td></tr>
<tr><td>Reserved</td><td>0x0008-0x00FF (8-255)</td><td>Reserved for future CTI use</td><td>Reserved</td></tr>
<tr><td>User defined</td><td>0x0100-0x1FFF (256-8191)</td><td>User defined</td><td>User defined</td></tr>
<tr><td>Exception</td><td>0x2000-0xAFFF (8192-45055)</td><td>Exception messages (see below)</td><td>Exception specific</td></tr>
<tr><td>AbortException</td><td>0x2329 (9001)</td><td>An Abort was processes</td><td>0</td></tr>
<tr><td>InterlockException</td><td>0x233D (9021)</td><td>Interlock was tripped</td><td>Interlock bit mask</td></tr>
<tr><td>CmdProcException</td><td>0xA678 (42616)</td><td>Command processing exception</td><td>Exception type code</td></tr>
<tr><td>Reserved</td><td>0xB000-0xFFFF (45056-65535)</td><td>Reserved for future CTI use</td><td>Reserved</td></tr>
</table>
<p>A CmdProcException is further refined by the puiPayloadLow value. These codes are defined in the following table:</p>
</td></tr>
</table>

Table 10: Command Processing Exception Codes

CmdProcException Type	puiPayloadLow Code	Description
FifoEmptyTimeout	0x232A (9002)	Command FIFO empty timeout
EventTimeout	0x232B (9003)	Event ISR timeout
BadOpcode	0x232C (9004)	Bad opcode
FirmwareBug	0x232D (9005)	Internal firmware bug
WriteDigitalBad	0x232E (9006)	WriteDigital bad argument
SetLaserPowerBad	0x232F (9007)	SetLaserPower bad argument
SetCorrectionTableBad	0x2330 (9008)	SetCorrectionTable bad argument
SetLaserPulseBad	0x2331 (9009)	SetLaserPulse bad argument
WaitForIOBad	0x2332 (9010)	WaitForIO bad argument
WaitForIOTimeout	0x2333 (9011)	WaitForIO command timeout
SetLaserStandbyBad	0x2334 (9012)	SetLaserStandby bad argument
CPLDTimeout	0x2335 (9013)	Communications timeout to the CPLD
LaserActiveTimeout	0x2336 (9014)	Timeout waiting for the laser to go active
SetMOTFOrientationBad	0x2337 (9015)	SetMotfDirection bad argument
EnableMotfBad	0x2338 (9016)	EnableMotf bad argument
SetLaserPulseWidthPctBad	0x2339 (9017)	SetLaserPulseWidthPct bad argument
SetLaserPulsePeriodPctBad	0x233A (9018)	SetLaserPulsePeriodPct bad argument
SetFieldOrientationBad	0x233B (9019)	SetFieldOrientation bad argument
ServoFault	0x233C (9020)	Servo fault detected
WriteAnalog	0x233E (9022)	WriteAnalog bad argument
OuputDrawList	0x2392 (9106)	OuputDrawList bad argument

6.4 Broadcast and Session API Error Codes

Errors returned by the Broadcast and Session API are defined in Table 11 below

Table 11: API Error Codes

Error Name	Code	Description
Success	0	Operation successful
Error_AccessDenied	1	TCP/IP networking access was denied
Error_NotConnected	3	Client is not connected to the server
Error_IllegalClientId	4	Internal error
Error_InvalidPersistState	5	Internal error
Error_ServerNameNotFound	8	Requested server name is not valid
Error_InvalidParameter	9	Bad parameter to a method call
Error_Network	10	TCP/IP networking error
Error_DataNotFound	11	Requested data file not found
Error_PathNotFound	12	Specified path does not exist
Error_Access	13	Access to server file system was denied
Error_LocalAccess	14	Access to the client file system was denied or Server is under control of a local pendant
Error_DataUnknown	15	XML data type is unknown
Error_EventHandling	16	Internal event processing error
Error_NotAvailable	17	Server is not currently available
Error_Aborting	19	Server is currently aborting
Error_Aborted	20	Server action was aborted
Error_Exception	23	Internal error
Error_Timeout	24	Requested action timed out
Error_NoData	25	The requested fixed data was empty
Error_DataExists	26	Destination file already exists and over-write not specified
Error_RemoteAccess	28	Server is already connected to a client
Error_StateError	29	Server is in an error state and unavailable
Error_BufferFull	31	Streaming data transmit buffer is full

7 Remote Control Protocol

There are three basic modes of operation for the EC1000:

1. LAN based streaming mode where job data is managed on a host computer and sent to the EC1000 for immediate execution
2. Local mode where an attached pendant is used to control the selection and execution of jobs stored locally
3. Remote mode where a LAN based supervisory interface can interact with the EC1000 and control all of the local mode functions

Remote mode is implemented as a text based messaging interface over a normal TCP/IP socket connection. Messages are sent to the EC1000 as strings terminated with a line-feed character. All messages sent to the EC1000 are acknowleged with a line-feed terminated string.

All read or *Get* functions can be executed concurrently with other activities the board may be performing, such as running jobs over the streaming interface. These functions would typically be associated with administrative functions such as examining passwords, networking parameters, job lists, etc. If modifications need to be made or if actual execution control is required via the remote control interface, then a client application must "request control" or ownership of the module via the protocol command *TakeHostControl*.

7.1 TCP/IP Interface

Remote control of the EC1000 can be established by any host computer that supports TCP/IP networking. This includes computers running Microsoft Windows, Linux, or other Unix derivitives. Communication with the board is established by opening a socket connection using the EC1000 IP address on port number 12500. The IP address can be learned by using the ILecBroadcast API to access the SysInfo data packets that are broadcast by the EC1000. Alternatively, if the EC1000 is configured with a static IP address, then broadcast monitoring is not required.

When a connection is established, the EC1000 transmits a "welcome banner". This string must be read from the socket before bi-directional communication can be established.

7.2 Protocol Specification

The following table defines the valid remote control commands and responses. Some commands take arguments. In such cases, the arguments are separated from the command and from each other by a ',' (comma) character. If commands yield responses that have multiple values, the values are comma separated.

Note that all commands are text strings and are expressed in the table enclosed in quotes (""). The quotation characters are NOT part of the command. This is also true for responses. Variable information is expressed as <variable> which is also a string.

Note also that all commands and arguments are case-sensitive.

Abort	
Purpose:	Stops the execution of a job
Implementation:	"Abort"
Parameters	
Returns:	"ACK" (command acknowledge)
Comments:	Immediately stops the execution of a running job and sets the *JobRunning* status
See also:	

CloseCOMPort	
Purpose:	Closes the serial I/O COM port on the EC1000
Implementation:	"CloseCOMPort,<port-ID>"
Parameters	<port-ID> (1 == pendant port, 2 == Laser serial port)
Returns:	"ACK" (command acknowledge)
Comments:	The COM port is closed and no longer available for serial I/O
See also:	*COMWriteLine, OpenCOMPort*

COMWriteLine	
Purpose:	Writes the string argument to the COM port on the EC1000.
Implementation:	"COMWriteLine,<port-ID>,<string>,<Timeout>"
Parameters	<port-ID> (1 == pendant port, 2 == Laser serial port) <string> (a string to be sent to the COM port) <Timeout> (time to wait in ms for a new-line terminated response)
Returns:	"<response string>" (command acknowledge) "ERROR_PORT_TIMEOUT" (if return string is not received before Timeout expires)
Comments:	This operation is intended to permit out-of-band commincation to serial port based automation devices or laser systems. The specified port-ID must have been opened with the command OpenCOMPort
See also:	*CloseCOMPort, OpenCOMPort*

ExecuteJobContinuous	
Purpose:	Starts the execution of a job and repeats it forever
Implementation:	"ExecuteJobContinuous,<job-name>"
Parameters	<job-name> (must be one of the jobs loaded with LoadFlashJob or LoadUSBJob)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have taken control of the EC1000 using the *TakeHostControl* command, and must have previously loaded a job from local (*LoadFlashJob*) or USB based (*LoadUSBJob*) Flash storage prior to issuing this command. Job execution will begin immediately. If job execution isrequired to be synchronous with an external input such as STRTMRK, then it should have been constructed with a <WaitForIO> instruction after the <BeginJob> instruction. At the completion of the job, the job will loop until an *Abort* command is received. This command returns as soon as the job is dispatched.
See also:	*TakeHostControl, GetJobStatus, Abort, LoadFlashJob, LoadUSBJob*

ExecuteJobOnce	
Purpose:	Starts the execution of a job one time
Implementation:	"ExecuteJobOnce,<job-name>"
Parameters	<job-name> (must be one of the jobs loaded with LoadFlashJob or LoadUSBJob)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have taken control of the EC1000 using the *TakeHostControl* command, and must have previously loaded a job from local (*LoadFlashJob*) or USB based (*LoadUSBJob*) Flash storage prior to issuing this command. Job execution will begin immediately without waiting unless itwas constructed with a <WaitFoIO> instruction. The job can be stoped at any time by issuing an *Abort* command. This command returns as soon as the job is dispatched.
See also:	*TakeHostControl, GetJobStatus, Abort, LoadFlashJob, LoadUSBJob*

GetAdminPIN	
Purpose:	Gets the current Administrator PIN (password)
Implementation:	"GetAdminPIN"
Parameters	
Returns:	<admin-pin> (Administrator PIN as a numeric string)
Comments:	The Admininstrator PIN is used with the Pendant interface to protect access to administrator functions
See also:	*SetAdminPIN, GetUserPIN, SetUserPIN*

GetDHCPMode	
Purpose:	Gets the current DHCP addressing mode
Implementation:	"GetDHCPMode"
Parameters	
Returns:	"Static" (Static IP addressing is used) "Autodetect" (Automatic DHCP based addressing is used)
Comments:	Static IP addressing is set using the *SetLocalIP*, *SetLocalGateway*, *SetSubnetMask* and *SetDHCPMode* command. The board must be reset before these setting take effect. Automatic IP addressing mode causes the EC1000 to request an IP address from a DHCP server when it boots up. If no server responds within a timeout period, the EC1000 automatically assigns itself an IP address in the range 169.254.xxx.yyy with a net-mask value of 255.255.0.0.
See also:	*SetLocalIP, SetLocalGateway, SetSubnetMask, SetDHCPMode*

GetFlashJobFileList	
Purpose:	Returns a comma separated list of job files located on the Flash file system located on the EC1000
Implementation:	"GetFlashJobFileList"
Parameters	
Returns:	<job-list> (a comma separated list of job names)
Comments:	Jobs are loaded into the EC1000 Flash file system through the use of the *ILecSession.saveJobData* method.
See also:	*ILecSession.saveJobData*

GetHostControlStatus	
Purpose:	Returns the current EC1000 control status of this remote control session
Implementation:	"GetHostControlStatus"
Parameters	
Returns:	"HOST_IN_CONTROL" (control has been granted to this session) "HOST_NOT_IN_CONTROL" (this session is not in exclusive control of the EC1000)
Comments:	
See also:	*TakehostControl, ReleaseHostControl*

GetHostInControl	
Purpose:	Returns the current host interface that has exclusive control of the EC1000
Implementation:	"GetHostInControl"
Parameters	
Returns:	"Pendant" (control has been granted to the pendant interface) "LANStream" (control has been granted to the streaming LAN interface) "LAN" (control has been granted to the LAN remote control interface)
Comments:	
See also:	*TakehostControl, ReleaseHostControl*

GetJobStatus	
Purpose:	Returns the status of the currently executing job
Implementation:	"GetJobStatus"
Parameters	
Returns:	"Idle" (no job is executing; a job may or may not be loaded) "Busy" (a job is executing)
Comments:	
See also:	

GetLocalGateway	
Purpose:	Returns the gateway IP address used by the EC1000 if in static IP addressing mode
Implementation:	"GetLocalGateway"
Parameters	
Returns:	<gateway-address> (in dot notation, e.g. 192.168.101.2)
Comments:	
See also:	*SetLocalGateway*

GetLocalIP	
Purpose:	Returns the IP address used by the EC1000 if in static IP addressing mode
Implementation:	"GetLocalIP"
Parameters	
Returns:	<static-IP-address> (in dot notation, e.g. 192.168.101.2)
Comments:	
See also:	*SetLocalIP*

GetNodeFriendlyName	
Purpose:	Returns the “friendly name” of the EC1000
Implementation:	“GetNodeFriendlyName”
Parameters	
Returns:	<friendly-name> (string representing the friendly name assigned to the EC1000)
Comments:	This corresponds to the tag <FriendlyName> in the AdminConfig file
See also:	*SetNodeFriendlyName*

GetRemoteIP	
Purpose:	Returns the IP address of the LAN stream host that has control of the EC1000
Implementation:	“GetRemoteIP”
Parameters	
Returns:	<remote-IP-address> (in dot notation, e.g. 192.168.101.2)
Comments:	If no host has control, the address “0.0.0.0” is returned
See also:	

GetSubnetMask	
Purpose:	Returns the subnet mask used by the EC1000 if in static IP addressing mode
Implementation:	“GetSubnetMask”
Parameters	
Returns:	<subnet-mask> (in dot notation, e.g. 255.255.255.0)
Comments:	
See also:	*SetSubnetMask*

GetUSBJobFileList	
Purpose:	Returns a comma separated list of job files located on the USB Flash file system attached to the EC1000
Implementation:	“GetUSBJobFileList”
Parameters	
Returns:	<job-list> (a comma separated list of job names)
Comments:	Jobs are loaded onto a USB Flash file system through the use of the *ILecSession.saveJobData* method.
See also:	*ILecSession.saveJobData*

GetUserPIN	
Purpose:	Gets the current User PIN (password)
Implementation:	“GetUserPIN”
Parameters	
Returns:	<user-pin> (User PIN as a numeric string)
Comments:	The User PIN is used with the Pendant interface to protect unauthorized access to EC1000 functions
See also:	*SetUserPIN, GetAdminPIN, SetAdminPIN*

HardwareReset	
Purpose:	Forces a hard reset of the EC1000
Implementation:	"HardwareReset"
Parameters	
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. The board will reboot as if power were just applied. Any IP addressing changes will be applied.
See also:	

LoadFlashJob	
Purpose:	Loads a job from the EC1000 resident Flash file system
Implementation:	"LoadFlashJob,<job-name>"
Parameters	<job-name> (the name of a job stored on the EC1000)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. The job name must include the extension as part of the name, e.g. "Circle.wlb"
See also:	*GetFlashJobList*

LoadHardwareDefaults	
Purpose:	Sets the current operating parameters of the EC1000 to their default values
Implementation:	"LoadHardwareDefaults"
Parameters	
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command.
See also:	

LoadUSBJob	
Purpose:	Loads a job from the USB Flash file system attached to the EC1000
Implementation:	"LoadUSBJob,<job-name>"
Parameters	<job-name> (the name of a job stored on the USB Flash file system device)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. The job name must include the extension as part of the name, e.g. "Circle.wlb"
See also:	*GetUSJobList*

OpenCOMPort	
Purpose:	Opens the specified serial I/O COM port on the EC1000
Implementation:	"OpenCOMPort,<port-ID>,<baud-rate>,<data-bits>,<parity>,<stop-bits>,<flow-control>"
Parameters	<port-ID> (1 == pendant port, 2 == Laser serial port <baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000}) <data-bits> (one of {5,6,7,8}) <parity> (one of {Even,Odd,None,Mark,Space}) <stop-bits> (one of {1,1.5,2}) <flow-control> (one of {None,XonXoff,CTS_RTS,DSR_DTR})
Returns:	"ACK" (command acknowledge)
Comments:	The specified COM port is opened and is available for serial I/O. This operation is intended to permit out-of-band commincation to serial port based automation devices or laser systems. A normal configuration might be specified as: OpenCOMPort,2,38400,8,None,1,None Only COM port-ID 1 has hardware flow control support.
See also:	*COMWriteLine, CloseCOMPort*

ReleaseHostControl	
Purpose:	Releases exclusive control of the EC1000 to the LANStream host interface
Implementation:	“ReleaseHostControl”
Parameters	
Returns:	“ACK” (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. Control is returned to the LANStream interface such that jobs may again be streamed to the EC1000 via that interface.
See also:	*TakeHostControl*

SetAdminPIN	
Purpose:	Sets the Administrator PIN (password)
Implementation:	“SetAdminPIN,<admin-pin>”
Parameters	<admin-pin> (new administrator PIN as a numeric string)
Returns:	“ACK” (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. The Admininstrator PIN is used with the Pendant interface to protect access to administrator functions
See also:	*SetAdminPIN, GetUserPIN, SetUserPIN*

SetDHCPMode	
Purpose:	Sets the DHCP addressing mode
Implementation:	“SetDHCPMode,<mode>”
Parameters	<mode> (“Static” or “Autodetect”)
Returns:	“ACK” (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. Static IP addressing parameters are set using the *SetLocalIP*, *SetLocalGateway*, and *SetSubnetMask* commands. The board must be reset before these setting take effect. Automatic IP addressing mode causes the EC1000 to request an IP address from a DHCP server when it boots up. If no server responds within a timeout period, the EC1000 automatically assigns itself an IP address in the range 169.254.xxx.yyy with a net-mask value of 255.255.0.0.
See also:	*SetLocalIP, SetLocalGateway, SetSubnetMask*

SetLocalGateway	
Purpose:	Sets the gateway IP address used by the EC1000 if in static IP addressing mode
Implementation:	“SetLocalGateway,<gateway-address>”
Parameters	<gateway-address> (in dot notation, e.g. 192.168.101.2)
Returns:	“ACK” (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. Other static IP addressing parameters are set using the *SetLocalIP* and *SetSubnetMask* commands. The board must be reset before these setting take effect.
See also:	*GetLocalGateway, SetLocalIP, SetSubnetMask, SetDHCPMode*

SetLocalIP	
Purpose:	Sets the IP address used by the EC1000 if in static IP addressing mode
Implementation:	“SetLocalIP,<IP-address>”
Parameters	<IP-address> (in dot notation, e.g. 192.168.101.200)
Returns:	“ACK” (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. Other static IP addressing parameters are set using the *SetLocalGateway* and *SetSubnetMask* commands. The board must be reset before these setting take effect.
See also:	*GetLocalIP, SetLocalGateway, SetSubnetMask, SetDHCPMode*

SetNodeFriendlyName	
Purpose:	Sets the "friendly name" of the EC1000
Implementation:	"SetNodeFriendlyName,<friendly-name>"
Parameters	<friendly-name> (string representing the friendly name assigned to the EC1000)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. This corresponds to the tag FriendlyName in the AdminConfig file
See also:	*GetNodeFriendlyName*

SetSubnetMask	
Purpose:	Sets the subnet mask used by the EC1000 if in static IP addressing mode
Implementation:	"SetSubnetMask,<mask>"
Parameters	<mask> (in dot notation, e.g. 255.255.255.0)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. Other static IP addressing parameters are set using the *SetLocalGateway* and *SetLocalIP* commands. The board must be reset before these setting take effect.
See also:	*GetSubnetMask, SetLocalGateway, SetLocalIP, SetDHCPMode*

• SetUserPIN	
Purpose:	Sets the Administrator PIN (password)
Implementation:	"SetUserPIN,<user-pin>"
Parameters	<user-pin> (new user PIN as a numeric string)
Returns:	"ACK" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. The User PIN is used with the Pendant interface to protect access to EC1000 functions
See also:	*SetUserPIN, GetAdminPIN, SetAdminPIN*

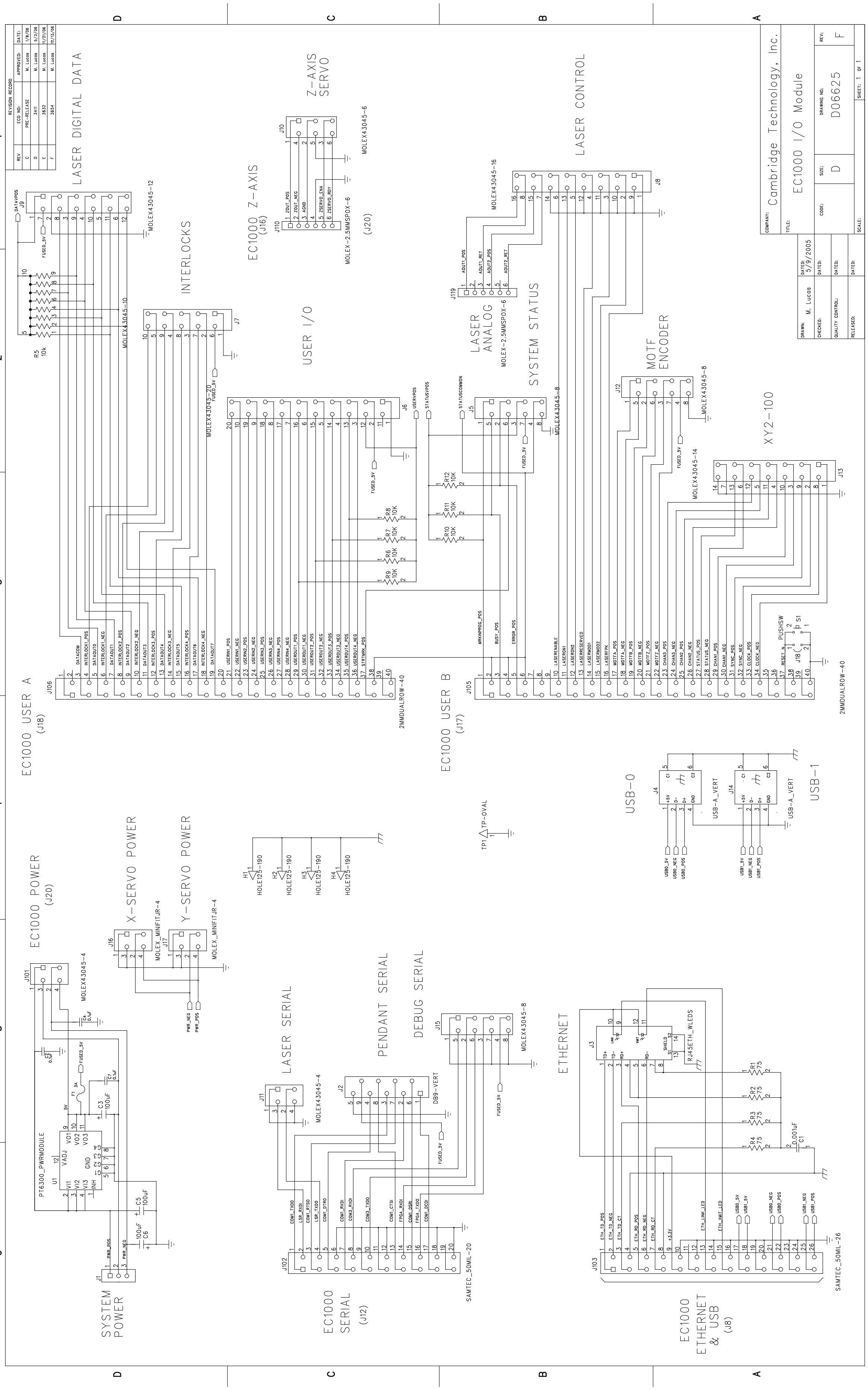
TakeHostControl	
Purpose:	Requests exclusive control of the EC1000
Implementation:	"TakeHostControl"
Parameters	
Returns:	"ACK" (command acknowledge)
Comments:	Exclusive control will not be granted if the EC1000 is currently executing a job. Use the *GetJobStatus* command to determine if the EC1000 is in a proper state before issuing this command.
See also:	*ReleaseHostControl, GetJobStatus*

7.3 Remote Control Error Codes

In certain cases, the response messages may be an error message rather than the expected ACK or return variable(s). The following table defines the possible error strings that may be returned.

Error String	Description
ERROR_ARGS	The command passed inappropriately formed arguments, or no argument if an argument was required
ERROR_COMMAND	The command was not recognized
ERROR_SOFTWARE	An internal software exception occured
ERROR_NOT_IN_HOST_CONTROL	The command required that exclusive control of the EC1000 be obtained (*TakeHostContro*)
ERROR_NO_FILES_FOUND	The named job file was not found
ERROR_NO_DRIVE	No USB disk drive was found
ERROR_JOB_BUSY	Command cannot execute a job is running
ERROR_CANNOT_OPEN_PORT	Cannot open the serial port
ERROR_PORT_NOT_OPEN	Serial port was not opened before the requested command
ERROR_PORT_TIMEOUT	The serial port timed-out waiting for input
ERROR_WRONG_HOST_TYPE	Serial port I/O is only allowed while the streaming LAN interface is in control
ERROR_PORT_NUMBER	An invalid COM port ID was specified
ERROR_CANNOT_CREATE_PORT	An error occured while trying to open a COM port for serial communications

Appendix A EC000-IO Schematic


REVISION RECORD
REV | ECO NO: | APPROVED: | DATE:
C | PRE-RELEASE | M. Lucas | 1/9/06
D | 3411 | M. Lucas | 5/3/06
E | 3632 | M. Lucas | 11/21/06
F | 3654 | M. Lucas | 12/13/06
LASER DIGITAL DATA
INTERLOCKS
EC1000 Z-AXIS (J16)
Z-AXIS SERVO
USER I/O
LASER CONTROL
LASER ANALOG
SYSTEM STATUS
MOTF ENCODER
XY2-100
EC1000 USER A (J18)
EC1000 USER B (J17)
USB-0
USB-1
EC1000 POWER (J20)
X-SERVO POWER
Y-SERVO POWER
LASER SERIAL
PENDANT SERIAL
DEBUG SERIAL
ETHERNET
SYSTEM POWER
EC1000 SERIAL (J12)
EC1000 ETHERNET & USB (J8)
PT6300_PWRMODULE
RJ45ETH_WLEDS
SAMTEC_50MIL-20
SAMTEC_50MIL-26
2MMDUALROW-40
COMPANY: Cambridge Technology, Inc.
TITLE: EC1000 I/O Module
CODE:
SIZE: D
DRAWING NO: D06625
REV: F
SCALE:
SHEET: 1 OF 1
DRAWN: M. Lucas
DATED: 5/9/2005
CHECKED:
DATED:
QUALITY CONTROL:
DATED:
RELEASED:
DATED: