EC1000^{TM} Controller

Advanced Laser Positioning & Control
for Laser Steering Systems

Software Reference Manual

Revision 2.3.0

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. P0900-0135   Rev. 2.3.0   18 Feb 2011

Table of Contents

Table of Contents

1 Introduction

1.1 General Notes

Cambridge Technology 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 covers the software programming API for the EC1000 Scan System Control Board only. Information on the EC1000 hardware can be obtained from the EC1000 Hardware Reference Manual. Additional detailed operational information is contained in *Appendix 1 Theory Of Operation*.

1.2.2 Revision History

REVISION	DATE	Changes from previous revision
2.0.0	6 Nov 2009	Extracted from P0900-0120, EC1000 OEM Integrators Manual Major revision for new language additions and reformatting to improve usability
2.0.1	30 Dec 2009	Corrected XML command for LaserPower
2.0.2	27 Jan 2010	Corrected units of LaserOnDelay and LaserOffDelay to usec from laserTicks Fixed typos in structured job data definitions
2.1.1	06 May 2010	Added support for Remote API over RS-232 Added new Remote API commands Added definition for error code 2 Added definition in the LaserConfig.xml file for setting the mode of the LASERON1 signal
2.1.2	07 June 2010	Added additional definitions in the LaserConfig.xml file and job parameters for setting the mode of the LASERON1 signal
2.2.0	10 Dec 2010	Fixed syntax error in WobbleEnable Clarified time units in laser configuration file Fixed bit mappings in LaserModeConfig in Job nad Config file XML RS-232 based Laser Initialization now supported Many other typos fixed. See change-bars.
2.3.0	18 Feb 2011	Added support for automatic job launching at boot time Added new API sendStreamData2() entry point for returning estimated execution time Added head parking function for dual head configurations Added 2D RT transform priority message

1.3 Obtaining Technical Assistance

If you encounter a problem:

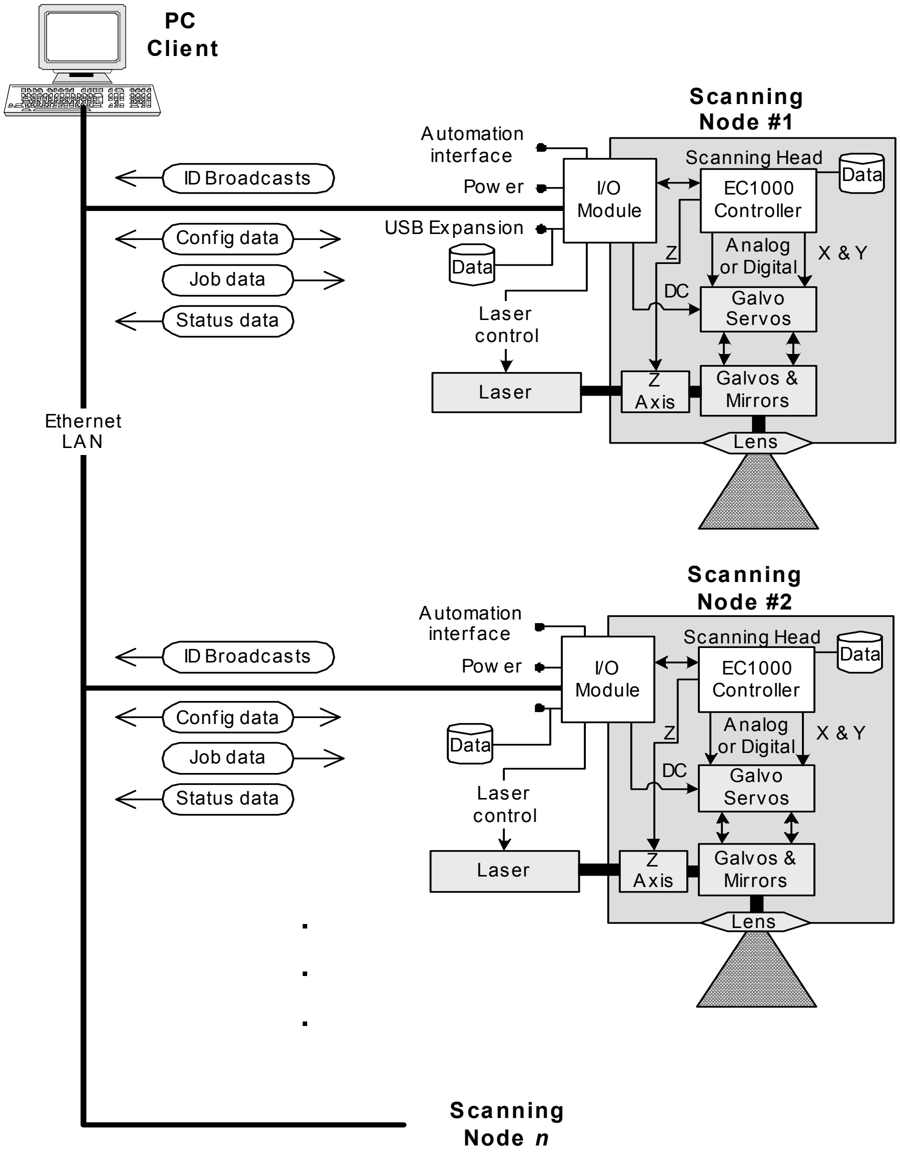
1. Review all of the information contained in this manual.
2. Consult your own internal people about the issue.
3. If you need further assistance, call Cambridge Technology, Monday through Friday, 9 A.M. until 4:30 P.M. (Eastern Standard Time) at 781-541-1600. You may also send an e-mail to SoftwareSupport@camtech.com.

2 EC1000 Product Introduction

2.1 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 and laser system. It is capable of controlling two scan-heads with up to three motion axes each with concurrent laser timing control. It also provides integrated synchronization I/O for connection to factory automation equipment.

Figure 1: EC1000 Control Board typical installation



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, laser control, external automation control,and diagnostic access. Optional touch-panel based operator interfaces are directly supported by the hardware and software.

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.

2.2 EC1000 Features

2.2.1 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
- one RS232 external automation control port
- integrated support for local LCD and touch panel interface

2.2.2 Software features

The EC1000 is designed with a client-server architectural model. 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 a TCP/IP as a transport mechanism over Ethernet.

To simplify integration with third-party application software a Microsoft Windows compatible Application Programming Interface (API) is provided. Three API formats are supported: .NET, COM, and Win32 DLL format. 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, Vista and Windows 7 operating systems
- administrative management of the EC1000 including scanning head configuration data
- automatic device recognition for any number of network attached controllers
- COM, .NET, and Win32 DLL access to all scanning functions using XML for parameter and command passing
- support for up to four lens correction files (65 x 65 data points) to correct for field distortions
- local user interface via serial pendant or attached LCD touch screen for fully stand-alone operation

2.3 Application Programming Interface

The host software Application Programming Interface (API) is implemented in Microsoft's C# language and is exposed as Windows .NET assemblies and as COM objects. It is also accessible via a bridge DLL that provides Win32 style access without the complexity of COM. These interfaces permit access from any suitable Microsoft Windows platform programming language such as Visual Basic, C++, C#, etc.

The DLLs and .tlb files that make up the COM interface are automatically installed and registered in the Window Registry by a setup installation program on the software distribution CD.

In Visual Studio Version 6 programming languages or National Instruments LabVIEW programming environment, the API can be accessed as COM objects that are imported into the IDE trough the use of the COM object browser, or by a more traditional Win32 style DLL calling interface. The COM 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.

2.3.1 Installation location

The DLLs that make up the API are installed in the following location on the installation drive, typically the C: drive:

C:\Program Files\CTI\EC1000\Client

The DLL names and their function are defined in Table 1:

Table 1: EC1000 API DLLs

DLL name	Function
LecBroadcast.dll, LecBroadcast.tlb	Contains the .NET and COM Broadcast API entry points
LecSession.dll, LecSession.tlb	Contains the .NET and COM Session API entry points
CommonLib.dll FTPClient.dll	Contains support functions for the API. Required for use.
EC1000Win32.dll, EC1000Win32.lib, EC1000Win32.h	Contains Win32 compatible entry points to the API. This is an alternate interface to the LecBroadcast and LecSession methods.
CTITelnet.dll, telnet.dll	Utility functions to support Telnet access to the EC1000. Private to CTI. Used by the firmware updater utility.
ECUtils.dll	Utility functions used by the demo programs. Not necessary for normal use.

2.3.2 API structure

The API is divided into three components:

1. the Broadcast API, used to identify EC1000 modules on the network
2. the Session API, used to transfer configuration and job data to and from a selected controller for real-time processing
3. the Remote Control API, used to provide simple ASCII character-string level control of an EC1000 that has been conditioned to be able to run previously prepared and locally stored marking jobs

For convenience, the API is defined using Visual Basic syntax. All functions return integer codes to indicate the success or failure of the operations. These codes are defined in section *5.7 Session API Error Codes*

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

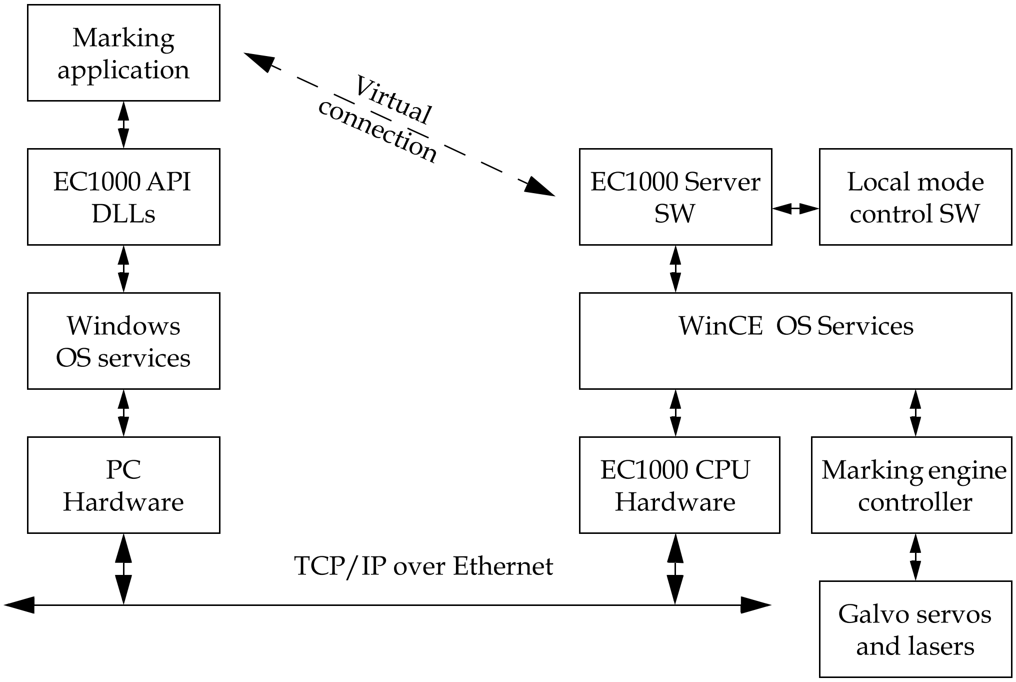
Sample programs illustrating the use of the API are located in the C:\Program Files\CTI\EC1000\Client\Sample Programs directory.

3 Software overview

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.

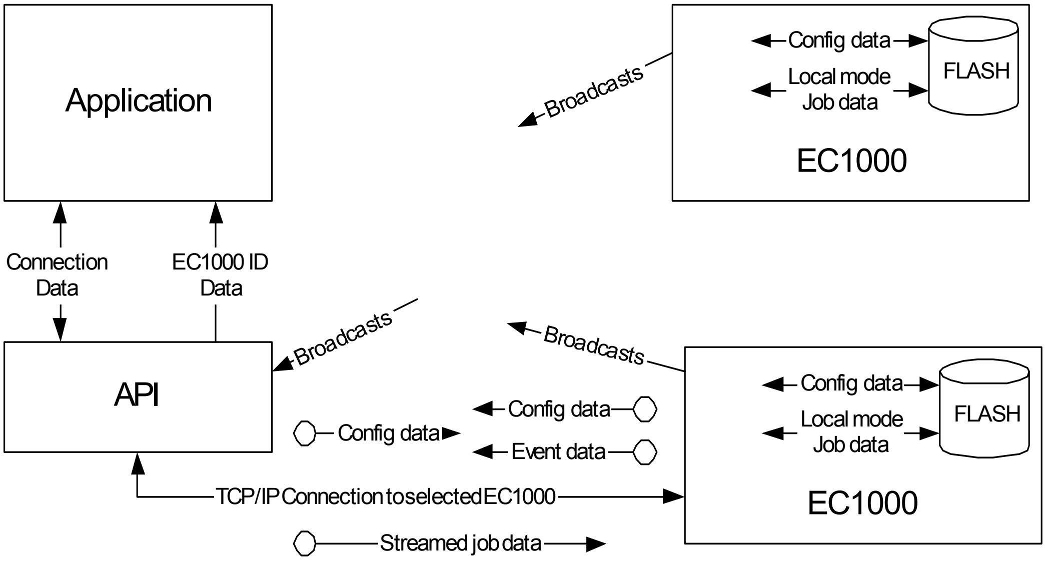
Job data is typically prepared using editor applications designed for that purpose. These applications may be custom software applications written by an OEM integrator, or one of several commercially available packages. These applications are hosted on a Microsoft Windows^{TM} based PC and interface to the EC1000 modules through the API DLLs. The DLLs takes care of establishing and maintaining communications with an EC1000, and provides a managed conduit for passing data to and from the controller. *Figure 2: Client-server architecture* illustrates this arrangement.

Figure 2: Client-server architecture



The EC1000 contains a fully integrated processor and Windows CE 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 API 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 3: EC1000 Software Data Flow* illustrates this relationship.

Figure 3: EC1000 Software Data Flow



A communications session 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 message propagation back to the application.

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.

3.1 The use of XML in the API

The API uses XML syntax for setting laser timing and scanner parameters, and for specifying motion vector sequences at any desired speed. XML is a standard text-based specification language used in many internet applications to represent data in a portable manner. Documentation on XML is available from many on-line sources.

Job commands and configuration data elements can take multiple arguments to specify their function. In addition, data may be numeric of several different types or text strings. Depending on the command, parameters may be passed as XML attributes or as token values. Lists of values are separated using a comma “,” or semi-colon “;”. Where lists of floating point values are passed, the semi-colon separator is prefered to avoid problems with internationalization of the comma character as a decimal place specifier. *Table 2: Sample XML statements* shows a few samples of how XML is used in the API.

Table 2: Sample XML statements

XML statement	Meaning
<set id=’JumpDelay’>200</set>	The “set” command is used to specify parameters that modify the behavior of a job when it is run
<MarkAbs>1000, 2000, 300</MarkAbs>	Draw a marking vector from the current position to the target location specified in whole numbers (normally bits units)
<JumpAbs>1.25; 15.5; 0.3</JumpAbs>	Jump from the current position to the target location specified in floating point numbers (could be mm or inch units). Note the use of the semi-colon separator
<LaserStandby laser=’1’ width=’10’ period=’200’ />	Set the standby laser modulation characteristics for the LASERMOD1 output to a pulse width of 10 laser timing ticks with a period of 200 laser timing ticks. This attribute style notation is used in the configuration files.
<set id=’LaserStandby’>1, 10, 200</set>	Equivalent to the previous example except this is the form used in a job.

Details of these statements and all others is contained in the following sections.

4 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 Section *5 Session API*.

4.1 Establishing a connection

To use the broadcast facility, a connection must be made to the BroadcastAPI using the following functions.

4.1.1 clientAttachBroadcast

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 // Error codes are defined in section *5.7 Session API Error Codes*
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 directory.
See also	ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerCount(), ILecBroadcast.getLecServerList(), ILecBroadcast.getBroadcastData()

4.1.2 clientDetachBroadcast

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 // Error codes are defined in section *5.7 Session API Error Codes*
Explanation	This method is used by a client application to terminate a connection to the broadcast mechanism of the EC1000.
See also	ILecBroadcast.clientAttachBroadcast()

4.2 Retrieving broadcast data

Several methods are provided to get information about network-attached EC1000 modules.

4.2.1 getLecServerCount

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 // Error codes are defined in section *5.7 Session API Error Codes*
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.
See also	ILecBroadcast.clientAttachBroadcast(), ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerList(), ILecBroadcast.getBroadcastData()

4.2.2 getLecServerList

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 // Error codes are defined in section *5.7 Session API Error Codes*
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	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

4.2.3 getBroadcastData

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 // Error codes are defined in section *5.7 Session API Error Codes*
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.
See also	ILecBroadcast.clientAttachBroadcast(), ILecBroadcast.clientDetachBroadcast(), ILecBroadcast.getLecServerCount(), ILecBroadcast.getLecServerList()

4.3 Broadcast data definitions

Both the Broadcast and Session APIs uses a data type code, see *Table 3: Broadcast data type codes*, 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.

Table 3: Broadcast data type codes

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 in *Table 4: Data type keys*.

Table 4: Data type keys

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 32-bit Floating Point range
BOOL	Boolean	true, false
HEX	Unsigned 16-bit integer	0x0000 <-> 0xFFFF

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

4.3.1 Broadcasted 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="24">Definition</td><td>XML Tag</td><td>XML Example Text</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td><Data type='SysInfoData' rev='1.0'></td><td></td><td></td></tr>
<tr><td>MSN</td><td><MSN>**EC1000-000005**</MSN></td><td>STR</td><td>Unique board manufacturing code.</td></tr>
<tr><td>PVer</td><td><PVer>**0420**</PVer></td><td>STR</td><td>Version of the platform software.</td></tr>
<tr><td>AVer</td><td><AVer>**2.0.0**</AVer></td><td>STR</td><td>Version of the EC1000 embedded server software.</td></tr>
<tr><td>ObjExtVer</td><td><ObjExtVer>**cti.1.0**</ObjExtVer></td><td>STR</td><td>Version of the on-board object extension library</td></tr>
<tr><td>FPGAFirmVer</td><td><FPGAFirmVer>**20090931**</FPGAFirmVer></td><td>STR</td><td>Version of the FPGA firmware that is loaded.</td></tr>
<tr><td>StateCode</td><td><StateCode>**1**</StateCode></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 *(future)*</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><LastError>**0**</LastError></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><FreeTempStorage>**7805**</FreeTempStorage></td><td>U32</td><td>The amount of free storage in non-persistent memory in Kilo Bytes.</td></tr>
<tr><td>PermStoragePath</td><td><PermStoragePath>**Disk**</PermStoragePath></td><td>STR</td><td>The path to the root of persistent memory.</td></tr>
<tr><td>FreePermStorage</td><td><FreePermStorage>**29616**</FreePermStorage></td><td>U32</td><td>The amount of free storage in persistent memory in Kbytes.</td></tr>
<tr><td>FreeUSBStorage</td><td><FreeUSBStorage>**100220**</FreeUSBStorage></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><MAC>**00:50:C2:4F:A0:00**</MAC></td><td>STR</td><td>Hardware address.</td></tr>
<tr><td>NetMask</td><td><NetMask>**255.255.255.0**</NetMask></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><NetAssign>**1**</NetAssign></td><td>I32</td><td>Network assignment is either manual, provided by DHCP, or provided by DNS.</td></tr>
<tr><td>IP</td><td><IP>**192.168.2.1**</IP></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><ConnectIP>**192.168.2.65**</ConnectIP></td><td>STR</td><td>The client IP address that is currently connected to EC1000.</td></tr>
<tr><td>FriendlyName</td><td><FriendlyName>**EC_Alpha**</FriendlyName></td><td>STR</td><td>Name used by EC1000.</td></tr>
<tr><td>ConnectJob</td><td><ConnectJob>**Hubble**</ConnectJob></td><td>STR</td><td>The job name that is currently marking.</td></tr>
<tr><td>Port</td><td><Port>**12200**</Port></td><td>U32</td><td>The network port currently in use by the Job Session.</td></tr>
<tr><td>HSN</td><td><HSN>**HEAD-0000023**</HSN></td><td>STR</td><td>Marking head serial number.</td></tr>
<tr><td></td><td></Data></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Explanation</td><td colspan="4">This data defines 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>

4.3.2 Broadcasted 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='1.1'>		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. This information is derived from the XY2-100 status return, bit 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>**false**</XTemp>	I16	This value is true if the X galvo servo indicates an over-temperature condition in the XY2-100 status word. Note that this feature is not supported by all galvo controllers.
	YTemp	<YTemp>**false**</YTemp>	I16	This value is true if the Y galvo servo indicates an over-temperature condition in the XY2-100 status word. Note that this feature is not supported by all galvo controllers.
	ContrlTemp	<ContrlTemp>**32**</ContrlTemp>	I16	The value of the temperature in Celsius of the EC1000 controller.
	XStatus	<XStatus>**0x31**</XStatus>	U8	Inverted low-byte from the XY2-100 status return. Note that this value is galvo servo-controller specific.
	YStatus	<YStatus>**0x31**</YStatus>	U8	Inverted low-byte from the XY2-100 status return. Note that this value is galvo servo-controller specific.
	XPower	<XPower>**true**</XPower>	U8	This value is true if any of the bits in the XStatus register are asserted. Note that this value is galvo servo-controller specific.
	YPower	<YPower>**true**</YPower>	U8	This value is true if any of the bits in the YStatus register are asserted. Note that this value is galvo servo-controller specific.
	Interlock	<Interlock>**0x4**</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] == JOBACTIVE, ERROR/NREADY, 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 encounters a <BeginJob> command. 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()			

5 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.

Concurrent access to multiple EC1000 on a network is supported by creating multiple EC1000 session objects and separately logging into each one. Only a single host application can log into an EC1000 at a time.

5.1 Access to EC1000 modules

5.1.1 loginSession

A session is 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 // Error codes are defined in section *5.7 Session API Error Codes*
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 directory.
See also	ILecSession.logoutSession(), ILecSession.requestFixedData(), ILecSession.sendFixedData(), ILecSession.sendSteamData(), ILecSession.sendPriorityData()

5.1.2 logoutSession

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

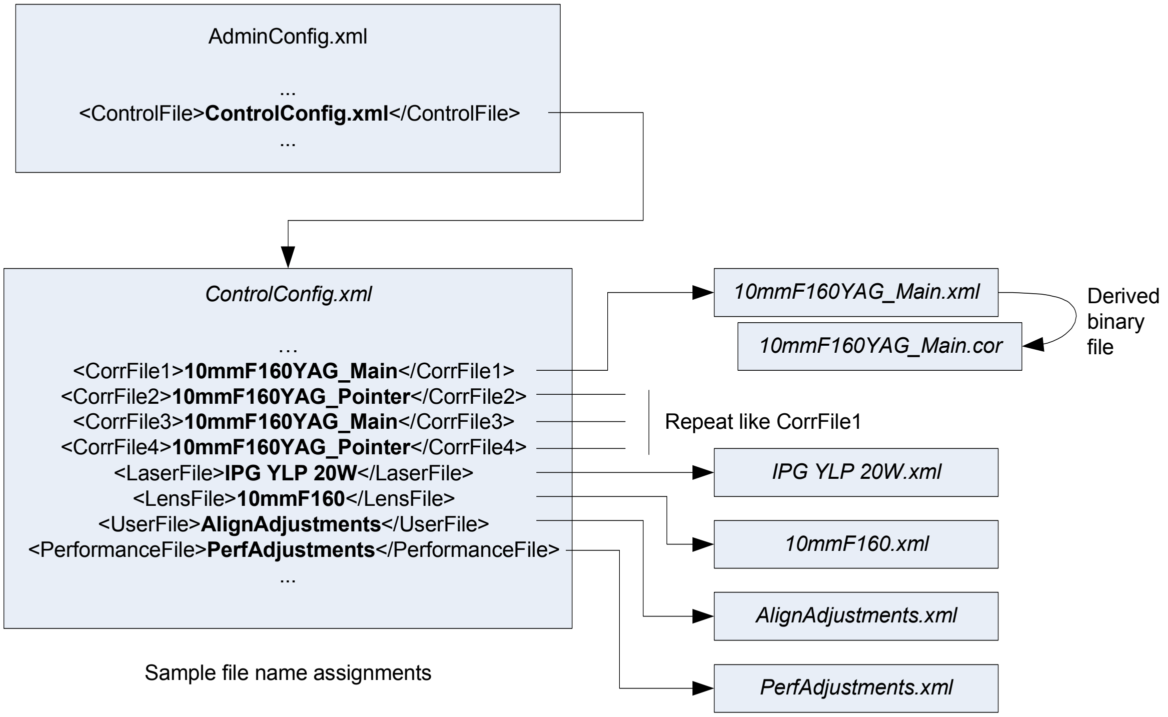
Command	ILecSession.logoutSession
Purpose	Disconnect an EC device session
Usage	ILecSession.logoutSession(ByVal puiTimeout As Unsigned Long // Duration for attempting call in seconds) As Unsigned Long // Error codes are defined in section *5.7 Session API Error Codes*
Explanation	When session communication 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 immediately aborted.
See also	ILecSession.loginSession()

5.2 Configuration data management

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 as "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 configuration data is defined in section *5.2.3 Configuration data definitions*.

Several XML data files make up the configuration data in a heirarchical relationship as shown in *Figure 4: EC1000 configuration file relationships*.

Figure 4: EC1000 configuration file relationships


AdminConfig.xml
...
<ControlFile>**ControlConfig.xml**</ControlFile>
...
ControlConfig.xml
…
<CorrFile1>**10mmF160YAG_Main**</CorrFile1>
<CorrFile2>**10mmF160YAG_Pointer**</CorrFile2>
<CorrFile3>**10mmF160YAG_Main**</CorrFile3>
<CorrFile4>**10mmF160YAG_Pointer**</CorrFile4>
<LaserFile>**IPG YLP 20W**</LaserFile>
<LensFile>**10mmF160**</LensFile>
<UserFile>**AlignAdjustments**</UserFile>
<PerformanceFile>**PerfAdjustments**</PerformanceFile>
...
Sample file name assignments
10mmF160YAG_Main.xml
10mmF160YAG_Main.cor
Derived binary file
Repeat like CorrFile1
IPG YLP 20W.xml
10mmF160.xml
AlignAdjustments.xml
PerfAdjustments.xml

5.2.1 requestFixedData

The content of the configuration files on the EC1000 is accesed by using the requestFixedData() method:

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 *Table 5: Fixed data type codes*.
	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 // Error codes are defined in section *5.7 Session API Error Codes*

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 section *5.2.3 Configuration data definitions*. The AdminConfig.xml data file (see *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.
See also	ILecSession.sendFixedData()

5.2.2 sendFixedData

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 // Error codes are defined in section *5.7 Session API Error Codes*

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 immediately used and also the next time the module is booted. An application should wait on an application message event "FixedDataProcessed" to be assured the updated data has been processed by the EC1000 and is ready for subsequent actions.
See also	ILecSession.requestFixedData

5.2.3 Configuration 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.

Table 5: Fixed data type codes

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 in *Table 4: Data type keys*.

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 hardware 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.

Some 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 shown first in the tables and identified with the heading “Host application initialization settings”. The properties are ignored by the controller at boot-up.

The other data in the tables identified with the heading “Hardware initialization settings” are used by the controller at boot-up to configure the laser control signals and other hardware features.

All of the configuration 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 defines the name of the controller configuration file which contains pointers to other files that define the configuration of the module.

Purpose	The administration configuration describes configurable properties of the EC device related to system administration			
	XML Tag	**XML Example Text**	**Type**	**Description**
Header	Data	<Data type="AdminData" rev="1.0">		AdminData identifier
Job data channel settings	DataChannel	<DataChannel>		Channel ID for the transmission of job data
	Port	<Port>**12200**</Port>	U32	The TCP/IP port number used to pass job and fixed data to and from the EC1000
	ControlFile	<ControlFile>**ControlConfig.xml**</ControlFile>	STR	File name of the controller config data
	LogFile	<LogFile></LogFile>	STR	Path to a logging file for DataChannel transactions. Used only for system debugging.
	EnableStreamToFile	<EnableStreamToFile>**False** </EnableStreamToFile>	BOOL	If True, streaming job data is sent to the <LogFile> Used only for system debugging.
	StreamFile	<StreamFile></StreamFile>	STR	Name of a file that will capture data streamed to the device. Used only for system debugging.
		</DataChannel>		End DataChannel
Priority data channel settings	PriorityChannel	<PriorityChannel>	U32	Channel ID for the transmission of priority messages
	Port	<Port>**12201**</Port>	U32	The TCP/IP port number used to pass priority command data to the EC1000
	LogFile	<LogFile></LogFile>	STR	Path to a logging file for PriorityChannel transactions. Used only for system debugging.
		</PriorityChannel>		End PriorityChannel
Event channel settings	EventChannel	<EventChannel>	U32	Channel ID for the transmission of event messages
	Port	<Port>**12202**</Port>	U32	The TCP/IP port number used to pass event data from the EC1000 back to the host
	LogFile	<LogFile></LogFile>	STR	Path to a logging file for EventChannel transactions. Used only for system debugging.
		</EventChannel>		End AliveChannel
Heartbeat channel settings	AliveChannel	<AliveChannel>	U32	Channel ID for the transmission of alive messages
	Port	<Port>**12203**</Port>	U32	The TCP/IP port number used to pass heart-beat information between the EC1000 and the host
	LogFile	<LogFile></LogFile>	STR	Path to a logging file for AliveChannel transactions. Used only for system debugging.
		</AliveChannel>		End AliveChannel
Broadcast channel settings	BroadcastChannel	<BroadcastChannel>		Channel ID for the transmission of broadcast msgs
	Address	<Address>**224.168.100.2**</Address>	STR	IP address used for broadcast messages
	Port	<Port>**11000**</Port>	U32	The port number used for broadcast messages
	Retransmit	<Retransmit type="**SysInfoData**" time="**5**" />	U32	Broadcast period for the SysInfoData packet (sec)
	Retransmit	<Retransmit type="**StatInfoData**" time="**5**" />	U32	Broadcast period for the StatInfoData packet (sec)
		</BroadcastChannel>		End BroadcastChannel
Misc. settings	Settings	<Settings>		Various configuration settings
	FriendlyName	<FriendlyName>**EC_Alpha**</FriendlyName>	STR	The friendly name given this system
	HeadSerialNumber	<HeadSerialNumber>**XYZ**</HeadSerialNumber>	STR	Serial number of the head assigned by the OEM
	LocalMode	<LocalMode>**false**</LocalMode>	BOOL	The controller is to operate in local stand-alone mode on power-up. Pendant interactions are required to enable network operations.
	BreakOK	<BreakOK>**false**</BreakOK>	BOOL	*Reserved for future use.*
	Client	<Client>**LANStream**</Client>	STR	Selects the primary interface for accepting control information. Valid clients are: LANStream: LAN based streaming job control Pendant: Pendant based local control LAN: LAN based remote control RS232: RS232 based remote control
	Pendant	<Pendant>**QTERMJ10**</Pendant>	STR	Selects the type of pendant device to be used when Client type is Pendant. Valid Pendant devices are: QTERMJ10: QSI Corp QTERM-J10 terminal. QTERMJ10 pendants must always be connected to the EC1000 COM1 port.
	PendantPort	<PendantPort>**COM1:**</PendantPort>	STR	Selects the COM port used for the pendant
	PendantPortSpeed	<PendantPortSpeed>**38400**</ PendantPortSpeed>	U32	Baud-rate for the pendant COM port

Purpose	The administration configuration describes configurable properties of the EC device related to system administration			
	XML Tag	**XML Example Text**	**Type**	**Description**
Misc. settings, continued	APIPort	<APIPort>**COM2:**</APIPort>	STR	Selects the COM port used for remote API access. If the port is not specified, then no serial remote API support is available.
	APIPortSpeed	<APIPortSpeed>**38400**</APIPortSpeed>	U32	Baud-rate for the API COM port
	MotionPort	<MotionPort></MotionPort>	STR	*(Future)* Selects the COM port used for external motion control access. If the port is not specified, then no serial motion control is available.
	MotionPortSpeed	<MotionPortSpeed>**38400**</MotionPortSpeed>	U32	*(Future)* Baud-rate for the motion control COM port
	LaserPort	<LaserPort>**COM3:**</LaserPort>	STR	Selects the COM port used for laser communication. If the port is not specified, then no serial laser control is available.
	LaserPortSpeed	<LaserPortSpeed>**38400**</LaserPortSpeed>	U32	Baud-rate for the laser COM port
	DebugPort	<DebugPort></LaserPort>	STR	If assigned to a free COM port, the firmware will print debug trace messages on that port. If the port is not specified, then no debug messagesare available.
	User	<User>**123456**</User>	STR	Password for accessing user-level pendant functions. Six numeric characters only.
	Admin	<Admin>**654321**</Admin>	STR	Password for accessing administrator-level pendant functions. Six numeric characters only.
	LoggingLevel	<LoggingLevel>**0**</LoggingLevel>	U32	Level of transaction logging to perform. Used only for system debugging.
	LogFileName	<LogFileName>**log.txt**</LogFileName>	STR	If a file name is used, logging messages will be saved to a file on the EC1000 if *LoggingLevel* is set greater than zero. Use only as instructed by CTI technical support as this can greatly decrease the performance of the EC1000.
		</Settings>		End Settings
Footer		</Data>		End AdminData
Explanation	These properties control how the EC1000 identifies itself and how it records tracing information about network transactions. All of these properties are used by the controller at boot-up.			
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()			

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**	**Description**
Header	Data	<Data type='ControlConfigData' rev='1.0'>		ControlConfigData identifier
Host application initialization settings				
For convenience of host application user parameter initialization	MotfCapable	<MotfCapable>**true**</MotfCapable>	BOOL	System is Mark On The Fly (MOTF) capable (true)
	MotfCalGain	<MotfCalGain>**1.0**</MotfCalGain>	FLT	MOTF digital gain factor. Used as a fine-tuning scalar adjustment of MotfEncoderCal.
Hardware initialization settings				
Configuration file redirection	CorrFile1	<CorrFile1>**CORRTAB1**</CorrFile1>	STR	The name of correction table 1 file
	CorrFile2	<CorrFile2>**CORRTAB2**</CorrFile2>	STR	The name of correction table 2 file
	CorrFile3	<CorrFile3>**CORRTAB3**</CorrFile3>	STR	The name of correction table 3 file
	CorrFile4	<CorrFile4>**CORRTAB4**</CorrFile4>	STR	The name of correction table 4 file
	LensFile	<LensFile>**LENSFILE2**</LensFile>	STR	The name of the lens configuration file
	LaserFile	<LaserFile>**LASERFILE4**</LaserFile>	STR	The name of the laser configuration file
	UserFile	<UserFile>**MYCONFIGFILE**</UserFile>	STR	The name of the user configuration file
	PerformanceFile	<PerformanceFile>**PADJUST**</ PerformanceFile>	STR	The name of the performance adjustments file

<table>
<tr><th rowspan="2">Purpose</th><th colspan="4">The controller configuration describes to the combination of components that make up the EC device</th></tr>
<tr><th>XML Tag</th><th>XML Example Text</th><th>Type</th><th>Description</th></tr>
<tr><td>Mark-on-the-fly configuration</td><td>MotfEncoderCal</td><td><MotfEncoderCal>**24.23**</MotfEncoderCal></td><td>FLT</td><td>MOTF calibration factor. Relates the encoder counts to laser positioning bits (bits/count)</td></tr>
<tr><td></td><td>MotfMode</td><td><MotfMode>**0**</MotfMode></td><td>U16</td><td>MOTF operational mode
0 - Use encoder
1 - Simulate encoder</td></tr>
<tr><td></td><td>MotfDirection</td><td><MotfDirection>**0**</MotfDirection></td><td>I16</td><td>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</td></tr>
<tr><td>Galvo control configuration</td><td>AxisDACRange</td><td><AxisDACRange>0x**1**</AxisDACRange></td><td>HEX</td><td>Bit-field encoded values that set the output voltage 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</td></tr>
<tr><td></td><td>ServoConfig</td><td><ServoConfig>**0x4**<ServoConfig></td><td>HEX</td><td>Servo interface configuration control (not supported by all galvo servos).. The value is bit-field encoded as follows:</td></tr>
<tr><td></td><td colspan="4">
<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></td><td>LaserPipelineDelay</td><td><LaserPipelineDelay>**450**</LaserPipelineDelay></td><td>U16</td><td>The time in laser timing ticks that all laser signals are delayed relative to micro-vector generation. This is used to compensate for the inherent delay in servo modules from when a command is applied to when the galvos actually respond.</td></tr>
<tr><td>Interlock configuration</td><td>IntlockConfig</td><td><IntlockConfig>**0x1707**</IntlockConfig></td><td>HEX</td><td>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 to generate 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></td></tr>
</table>

Purpose	The controller configuration describes to the combination of components that make up the EC device			
	XML Tag	**XML Example Text**	**Type**	**Description**
Post bootup initialization	<StartupJob>	<StartupJob>HWInit.job</StartupJob>	STR	Name of a locally stored job to run after the controller boots up. Jobs can be Rev 1.0 style (.wlb), Rev 2.0 style (.job), or ScanMaster style (.lsj)
Footer		</Data>		End ControlConfigData
Explanation	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.			
See Also	ILecSession.requestFixedData(), ILecSession.sendFixedData()			

Laser configuration

Purpose	The laser configuration defines the properties of the laser in use with the EC device			
	XML Tag	**XML Example Text**	**Type**	**Description**
Header	Data	<Data type=’LaserConfigData’ rev=’2.0’>		LaserConfigData identifier and revision
Host application initialization settings (not required or used by the hardware)				
For convenience of host application user-parameter initialization	LsrName	<LsrName>**IPC002**</LsrName>	STR	The name of the laser
	LsrType	<LsrType>**1**</LsrType>	U16	Application definable value to identify a laser type
	FixedFreq	<FixedFreq>**true**</FixedFreq>	BOOL	Laser is only capable of a fixed frequency setting (true), or capable of variable frequency settings (false)
	FixedPW	<FixedPW>**true**</FixedPW>	BOOL	Laser is only capable of a fixed pulse width setting (true), or capable of variable pulse width settings (false)
	FixedWatts	<FixedWatts>**true**</FixedWatts>	BOOL	Laser is only capable of a fixed output power setting (true), or capable of variable output power settings (false)
	WattsUnits	<WattsUnits>**true**</WattsUnits>	BOOL	Laser power units are in Watts (true), or % duty-cycle (false)
	Pulse	<Pulse min=’**2**’ max=’**65535**’/>	U16	pulse width range supported by the laser (μsec)
	Bits	<Bits min=’**0**’ max=’**255**’/>	U16	Binary value range for lasers with digital power control
	ExtPwrCtrl	<ExtPwrCtrl>**false**</ExtPwrCtrl>	BOOL	Laser power is controllable via an external knob (true)
	UseExtPwrCtrl	<UseExtPwrCtrl>**false**</UseExtPwrCtrl>	BOOL	Application is configured to use external power control (true)
	VisPtr	<VisPtr>**false**</VisPtr>	BOOL	Laser has a visible pointer integrated into it (true)
	Duty	<Duty min=’**2**’ max=’**90**’/>	U16	Duty cycle range of the laser pulses (%)
	Freq	<Freq min=’**2**’ max=’**100**’/>	U16	Pulse frequency range sustainable by the laser (KHz)
	Watts	<Watts min=’**1**’ max=’**15**’/>	U16	Wattage range producible by the laser
	Volts	<Volts min=’**1**’ max=’**10**’/>	U16	Analog power level voltage range sustainable by the laser. The EC1000 is capable of 0-10 Volts output.
	Interlock	<Interlock>**IPCIntlocks.txt**</Interlock>	STR	The name of a file on the host platform that contains instructions on how to clear an interlock break
LaserConfigData rev=’2.0’ compatible hardware initialization settings. Use for new designs. **Do not use if rev=’1.0’ parameters are used (see below)**				
Configuration settings	LaserModeConfig	<LaserModeConfig>**0x140**</LaserModeConfig>	U16	Set the laser configuration using a bit mask encoded as follows:

Bit function	**Hex Bit Value**	**Definition**
LASERON1 polarity	0x0001	0=active high, 1=active low
LASERON2 polarity	0x0002	0=active high, 1=active low
LASERMOD1 polarity	0x0008	0=active high, 1=active low
LASERMOD2 polarity	0x0010	0=active high, 1=active low
LASERFPK polarity	0x0020	0=active high, 1=active low
LASERENABLE polarity	0x0040	0=active high, 1=active low
LSRPWRDOUT polarity	0x0080	0=active high, 1=active low
Laser activate	0x0100	1=activate (enable) laser output signals
Laser Power Port mode	0x0200	Set the mode of the digital laser power port 0=8-bit mode, 1=7-bit mode (LSB used as strobe)
LASERON2 configuration	0x0800 & 0x0400	Sets the mode of operation of LASERON2 0 - LASERON2 == !LASERON1 1 - LASERON2 == LASERON1 & !LasersEnabled 2 - LASERON2 == !LasersEnabled 3 - LASERON2 == Asserted all of the time
Laser Power Port	0x1000	0=8-bit digital power port, 1=analog output A1
LASERON1 configuration	0x2000	0=Gating signal, 1=Modulation signal if 8-bit digital power port bit 7 is also set

Purpose	The laser configuration defines the properties of the laser in use with the EC device			
	XML Tag	**XML Example Text**	**Type**	**Description**
Modulation control	LaserTiming	<LaserTiming>**50**</LaserTiming>	U16	(20 nsec increments) The number of 20ns intervals that make up a laser timing "tick"
	LaserEnableDelay	<LaserEnableDelay>**10** </LaserEnableDelay>	U16	(msec) The time require for enabling the laser prior to actual use Set the time that the signal LASERENABLE is asserted prior to a marking operation.
	LaserEnableTimeout	<LaserEnableTimeout>**20** </LaserEnableTimeout>	U16	(msec) 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 LaserEnableDelay.
	LaserModDelay	<LaserModDelay>**20**</LaserModDelay>	U16	(µsec) The time from the assertion of LASERON to the emission of laser pulses.
	LaserFPK	<LaserFPK position=’**0**’ width=’**10**’ />	I16	(µsec) Sets the ‘position’ of the LASERFPK signal relative to the LASERON signal, and the ‘width’ of the FPK pulse.
	LaserStandby	<LaserStandby laser=’1’ width=’**1**’ period=’**200**’ /> <LaserStandby laser=’2’ width=’**1**’ period=’**200**’ />	U16	(µsec) Sets the modulation characteristics of the LASERMOD1 and LASERMOD2 signals for the idle or non-lasing interval. ‘period’ must be the same for both lasers.
Power control	LaserPowerDelay	<LaserPowerDelay>**100** </LaserPowerDelay>	U16	(msec) The time required after laser power is changed until the laser power has settled. Used when constructing jobs that manipulate the laser power.
LaserConfigData rev=’1.0 and rev=’2.0’ compatible hardware initialization settings				
Power control	InitAnalog	<InitAnalog port=’0’ value=’0’ />	U16	Sets the initial value of analog output 1 (port 0). Value ranges from 0 to 4095 corresponding to 0 to 10V.
	InitAnalog	<InitAnalog port=’1’ value=’2048’ />	U16	Sets the initial value of analog output 2 (port 1). Value ranges from 0 to 4095 corresponding to 0 to 10V.
	InitDigital	<InitDigital port=’102’ value=’27’ />	U16	Sets the initial value of the 8-bit Digital laser power port. Port number 102 selects this port.. Values range from 0 to 255
	CorrTable	<CorrTable> <Entry>**0**</Entry> <Entry>**1**</Entry> . . . <Entry>**255**</Entry> </CorrTable>	I8	*(Future)* 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.
Miscellaneous initialization	InitLaser	<InitLaser>**false**</InitLaser>	BOOL	Laser requires explicit initialization via serial communication (true)
	InitType	<InitType>**0**</InitType>	U16	Laser communications type: 0 = RS-232 Serial, 1 = Ethernet (*Future*)
	InitStrDelim	<InitStrDelim>**","**</InitStrDelim>	CHR	Delimiter character separating command and argument tokens in the InitString
	InitStrEOL	<InitStrEOL>**"\n"**</InitStrEOL>	CHR	Line termination character used by the laser command interpreter
	InitStrings	<InitStrings> <InitString>**ab**</InitString> <InitString>**cd**</InitString> <InitString>**ef**</InitString> </InitStrings>	STR	A list of initialization strings to be sent to the laser. The list may be arbitrarily long
	DeinitStrings	<DeinitStrings> <DeinitString>**zy**</DeinitString> <DeinitString>**xw**</DeinitString> <DeinitString>**vu**</DeinitString> </DeinitStrings>	STR	A list of de-initialization strings to be sent to the laser. The list may be arbitrarily long

Purpose	The laser configuration defines the properties of the laser in use with the EC device			
	XML Tag	**XML Example Text**	**Type**	**Description**
	LaserConfigData rev='1.0' backwards compatible hardware initialization settings			
	NOTE: These settings are being deprecated and not recommended for future use. Do not use if new rev='2.0' parameters are used (see above)			
Configuration settings	LENAHigh	<LENAHigh>**false**</LENAHigh>	BOOL	LASERENABLE signal is active high (true) or active low (false)
	LONHigh	<LONHigh>**false**</LONHigh>	BOOL	LASERON1 signal is active high (true) or active low (false)
	LON2High	<LON2High>**false**</LON2High>	BOOL	LASERON2 signal is active high (true) or active low (false)
	LMOD1High	<LMOD1High>**false**</LMOD1High>	BOOL	LASERMOD1 signal is active high (true) or active low (false)
	LMOD2High	<LMOD2High>**false**</LMOD2High>	BOOL	LASERMOD2 signal is active high (true) or active low (false)
	LFPKHigh	<LFPKHigh>**false**</LFPKHigh>	BOOL	LASERFPK signals is active high (true) or active low (false)
	LON1Cfg	<LON1Cfg>**0**</LON1Cfg>	U16	Sets the mode of operation of LASERON1 0 - Gating signal. Is asserted when the laser is expected to be turned on. 1 - Modulation signal. Is modulated when the laser is expected to be turned on with the same properties as the LASERMOD1 signal, but only if bit 7 is set in the 8-bit digital laser port. Is unasserted with no modulation when the laser is expected to be turned off.
	LON2Cfg	<LON2Cfg>**1**</LON2Cfg>	U16	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>
Modulation control	LsrTiming	<LsrTiming>**50**</LsrTiming>	U16	(20ns increments) The number of 20ns intervals that make up a laser timing "tick"
	LsrEnaDly	<LsrEnaDly>**8**</LsrEnaDly>	U16	(msec) The time require for enabling the laser prior to actual use. Set the time that the signal LASERENABLE is asserted prior to a marking operation.
	LsrEnaTmo	<LsrEnaTmo>**10**</LsrEnaTmo>	U16	(msec) 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	(µsec) The time from the assertion of LASERON to the emission of laser pulses.
	FpsPos	<FpsPos>**30**</FpsPos>	U16	(µsec) The time between the assertion of the LASERFPK signal and the generation of LASERMOD pulses.
	FpsWidth	<FpsWidth>**0**</FpsWidth>	U16	(µsec) The width of the LASERFPK pulse.
	TickleWidth1	<TickleWidth1>**5**</TickleWidth1>	U16	(µsec) The width of the LASERMOD1 pulses during standby
	TickleFreq1	<TickleFreq1>**1**</TickleFreq1>	U16	(KHz) The frequency of the LASERMOD1 pulses during standby
	TickleWidth2	<TickleWidth2>**5**</TickleWidth2>	U16	(µsec) The width of the LASERMOD2 pulses during standby
	TickleFreq2	<TickleFreq2>**1**</TickleFreq2>	U16	(KHz) *(Future)* The frequency of the LASERMOD2 pulses during standby. Currently TickleFreq1 and TickleFreq2 must be set equal to each other
Power control	LsrPwrMode	<LsrPwrMode>**8bit**</LsrPwrMode>	STR	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. In 7bit mode 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
	LsrPwrDly	<LsrPwrDly>**100**</LsrPwrDly>	U16	(msec) The time required after laser power is changed until the laser power has settled. Used when constructing jobs that manipulate the laser power.
	LsrPwrPort	<LsrPwrPort>**0**<LsrPwrPort>	U16	Sets which port to assign the job command <LaserPower> values to: 0 - 8-bit digital port 1 - Analog port 0)
Footer		</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**	**Description**
Header	Data	<Data type=’LensConfigData’ rev=’1.1’>		LensConfigData identifier
Host application initialization settings. Not used by the hardware.				
For convenience of host application user parameter initialization	LensName	<LensName>**S4LFT0163**</LensName>	STR	Used by the head integrator to identify a particular lens model
	CalFlag	<CalFlag>**false**</CalFlag>	BOOL	Used by an application to indicate that this lens can be calibrated.
	ZMode	<ZMode>**0**</ZMode>	U16	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.

	FocalLen	<FocalLen>**163**</FocalLen>	U32	Focal length of the lens (mm)
	Aperture	<Aperture>**15**</Aperture>	U32	Laser beam diameter entering the lens (mm)
Hardware initialization settings				
Calibration factors	KFactor	<KFactor>**500**</KFactor>	U32	Scale factor relating the X galvo command signals to the distance traveled by the laser (bits/mm)
	YKFactor	<YKFactor>**500**</YKFactor>	U32	Scale factor relating the Y galvo command signals to the distance traveled by the laser (bits/mm)
	ZKFactor	<ZKFactor>**1100**</ZKFactor>	U32	Scale factor relating the Z (focus) actuator command signals to the focal plane displacement (bits/mm)
Correction table adjustments	Tbl1XOff	<Tbl1XOff>**0**</Tbl1XOff>	I16	X axis offset to be applied to correction table 1 (bits)
	Tbl1YOff	<Tbl1YOff>**0**</Tbl1YOff>	I16	Y axis offset to be applied to correction table 1 (bits)
	Tbl1XGain	<Tbl1XGain>**1.0**</Tbl1XGain>	FLT	X axis gain to be applied to correction table 1
	Tbl1YGain	<Tbl1YGain>**1.0**</Tbl1YGain>	FLT	Y axis gain to be applied to correction table 1
	Tbl1Rotation	<Tbl1Rotation>**0.0**</Tbl1Rotation>	FLT	Field rotation to be applied to correction table 1 (degrees)
	Tbl2XOff	<Tbl2XOff>**0**</Tbl2XOff>	I16	X axis offset to be applied to correction table 2 (bits)
	Tbl2YOff	<Tbl2YOff>**0**</Tbl2YOff>	I16	Y axis offset to be applied to correction table 2 (bits)
	Tbl2XGain	<Tbl2XGain>**1.0**</Tbl2XGain>	FLT	X axis gain to be applied to correction table 2
	Tbl2YGain	<Tbl2YGain>**1.0**</Tbl2YGain>	FLT	Y axis gain to be applied to correction table 2
	Tbl2Rotation	<Tbl2Rotation>**0.0**</Tbl2Rotation>	FLT	Field rotation to be applied to correction table 2
	Tbl3XOff	<Tbl3XOff>**0**</Tbl3XOff>	I16	X axis offset to be applied to correction table 3 (bits)
	Tbl3YOff	<Tbl3YOff>**0**</Tbl3YOff>	I16	Y axis offset to be applied to correction table 3 (bits)
	Tbl3XGain	<Tbl3XGain>**1.0**</Tbl3XGain>	FLT	X axis gain to be applied to correction table 3
	Tbl3YGain	<Tbl3YGain>**1.0**</Tbl3YGain>	FLT	Y axis gain to be applied to correction table 3
	Tbl3Rotation	<Tbl3Rotation>**0.0**</Tbl3Rotation>	FLT	Field rotation to be applied to correction table 3 (degrees)
	Tbl4XOff	<Tbl4XOff>**0**</Tbl4XOff>	I16	X axis offset to be applied to correction table 4 (bits)
	Tbl4YOff	<Tbl4YOff>**0**</Tbl4YOff>	I16	Y axis offset to be applied to correction table 4 (bits)
	Tbl4XGain	<Tbl4XGain>**1.0**</Tbl4XGain>	FLT	X axis gain to be applied to correction table 4
	Tbl4YGain	<Tbl4YGain>**1.0**</Tbl4YGain>	FLT	Y axis gain to be applied to correction table 4
	Tbl4Rotation	<Tbl4Rotation>**0.0**</Tbl4Rotation>	FLT	Field rotation to be applied to correction table 4
Footer		</Data>		End LensConfigData
Explanation	These values are normally set by the integrator and not intended to be altered by a marking application. Note that the Tbl{1,2,3,4} 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			
	XML Tag	**XML Example Text**	**Type**	**Description**
Header	Data	<Data type=’CorrTableData’ rev=’1.0’>		CorrTableData identifier
Hardware initialization settings				
Correction table data	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).
Footer		</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 Configuration table contains values that are completely under the control of a marking application			
	XML Tag	**XML Example Text**	**Type**	**Description**
Header	Data	<Data type=’UserConfigData’ rev=’1.0’>		UserConfigData identifier
Optional host application initialization settings (Not used by the hardware)				
General purpose user variables	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
Hardware initialization settings (Required)				
Global adjustment to all correction tables	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
Footer		</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 persistent across reboots of the controller. It is up to the application to interpret the UserVar data as required.			

Performance adjustments

Purpose	The Performance Adjustments table contains values that are used to adjust job parameters while the job is executing			
	XML Tag	**XML Example Text**	**Type**	**Description**
Header	Data	<Data type='PerformanceMatrixData' rev='1.0'>		PerformanceMatrixData identifier
Hardware initialization settings				
Adjustments made to system parameters at run-time	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)
Footer		</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()			

5.3 Marking job specification

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.

sendStreamData() is non-blocking in the sense that it returns as soon as the data is passed to the downstream communications system for transfer 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.

Job data passed to the EC1000 remains in vector format until it reaches the real-time marking engine controller. Only then is it converted to time-domain command data and passed to the laser galvo controllers.

5.3.1 Job data types

The sreaming data interface can send several types of data:

Rev. 1.0 compatible data (defined with the attribute rev='1.0', see next section):

1. JobData (standard) - this is data that represents a marking job using the XML based job definition language described in the next section. This job data is executed immediately in the sequence it is sent through the interface.
2. LoopJobData (**Deprecated in favor of Rev 2.0 constructs**) - this is job data in the exact same XML format as Standard Job Data, but the job is executed infinitely in a loop. It is possible to send multiple Looping Job Data segments, one after the other, each with its own iteration argument. In this case, the EC1000 infinitely loops all of the segments in succession in the order they are received. As each segment is processed, that segment will be looped the number of times specified by the "loops" argument for that segment. This type of job can be stopped at any time by sending an Abort:Job priority message, or Paused/Resumed using priority messages.
3. CorrTableData - this data is in the same format at the correction table XML definition. Correction table data sent this way does not persist though an EC1000 power cycle.

Rev. 2.0 compatible data (defined with the attribute rev='2.0'):

4. JobData (structured) - this is data that uses XML constructs to group related job instructions together into a segment that can be loaded to the board once and referred to multiple times via a separate sequence definition. A sequence definition construct permits the ordering of execution and iteration of pre-loaded segments. Structured JobData is a super-set of the functionality of LoopJobData and is the preferred approach for creating iterative jobs. If job structing is not required for iteration control then the constructs can be ommited and the data will be streamed as in Rev 1.0.

5.3.2 Job data definition

Job data contains both action commands that direct the marking engine to perform specific operations, and parametric data that affects how the EC1000 hardware behaves. Parameter commands do not cause any action, but modify the behavior of subsequent action commands. To minimize the number of XML identifier tags to express a job, the XML definition make use of two types of constructs. All action 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=’JumpDelay’>**150**</set>	The parameter ’JumpDelay’ is set to 150µ sec
<set id=’MarkDelay’>**150**</set>	The parameter ’MarkDelay’ is set to 150µ sec
<set id=’PolyDelay’>**50**</set>	The parameter ’PolyDelay’ is set to 150µ sec
<set id=’LaserTiming’>50</set>	Set the laser time base tick to 50 20nsec periods (1usec)
<<set id=’LaserOnDelay’>**75**</set>	The parameter ’LaserOnDelay’ is set to 75 laser timing ticks
<set id=’LaserOffDelay’>**100**</set>	The parameter ’LaserOffDelay’ is set to 100 laser timing ticks
<set id=’LaserPulse’>**1,10,20**</set>	Set the modulation of LASERMOD1 to a pulse width of 10 laser timing ticks with a period of 20 laser timing ticks
<Set id=’JumpSpeed’>**10,30**</Set>	The parameter ’JumpSpeed’ is set to 30 bits per each 10µ sec update period
<Set id=’MarkSpeed’>**10,10**</Set>	The parameter ’MarkSpeed’ is set to 10 bits per each 10µ 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

5.3.3 Job type specification

The job type is defined in the header section of the job XML preceding the job commands.

XML Text	Description
<Data type=’JobData’ rev=’1.0’>	Standard Job Data type declaration
<Data type=’LoopJobData’ rev=’1.0’ loops=’100’> (NOTE: Deprecated in favor of Rev 2.0 constructs)	Looping Job Data type declaration. The *loops* attribute specifies the number of times to iterate the job data relative to other LoopJobData segments in use.
<Data type=’CorrTableData’ rev=’1.0’>	Correction Table data. See definitions in that section.
<Data type=’JobData’ rev=’2.0’>	Standard Structured Job Data type declaration

5.3.4 Job parameters and commands

Jobs are made up of parameter definitions and action commands. Parameters are defined using the *Set* tag. Multiple values for parameters are expressed in a comma separated list. Commands are represented by a key-word and one or more arguments in a list. Parameters and commands are grouped by function in the following sections.

User units conversion parameters.

Param Identifier	Description	Arguments	Units	Min	Max
Units	Sets to units of the motion commands. This command affects the units of all motion commands that reference a coordinate or offset. Example: <set id=’Units’>**2**</set>	Units defines the conversion ratio used to map a motion related command *value* from user units to internally required “bits” units. This command assumes that appropriate CalFactor values have been set using the commands <set id=’XYCalFactor’> and <set id=’ZCalFactor’>. The default is 0 (bits). Appropriate values are: 0 - bits (1:1) 1 - mm (value * CalFactor) 2 - inch (value * 25.4 * CalFactor) 3 - mils ((value/1000) * 25.4 * CalFactor)	N/A	0	3

Param Identifier	Description	Arguments	Units	Min	Max
XYCalFactor	Sets the X & Y axis calibration factor used in converting coordinate system units. Example: <set id='XYCalFactor'>**500**</set>	CalFactor is used as a multiplier in converting user motion command units into the internally required "bits" units. The actual ratio is defined per the <set id'=Units'> command.	bits/ mm	0	HW practical limit
ZCalFactor	Sets the Z axis calibration factor used in converting coordinate system units. Example: <set id='ZCalFactor'>**125**</set>	CalFactor units are bits/mm	bits/ mm	0	HW practical limit

Motion control parameters.

Param Identifier	Description	Arguments	Units	Min	Max
JumpDelay	Set the delay used at the end of a jump command. Example: <set id='JumpDelay'>**150**</set>	duration – length of time to delay	µsec	0	
JumpSpeed (Two argument syntax)	Set the jump speed of the laser. The parameters are normally derived from an application speed setting expressed as mm/sec, bits/msec or some other appropriate ratio. Example: <set id='JumpSpeed'>**10,30**</set>	stepTime – the duration between each micro-step, i.e. how often the galvo position command is update. Can be set in 1µsec increments	µsec	10	65535
		stepSize – the distance traveled for each micro-step. If stepTime is absent, then this argument is interpreted as the vector speed in user-units/second	bits	1 or min user-units/ sec	65535 or max user-units/ sec
JumpSpeed (Single argument syntax)	If invoked with a single argument, the value specified with the command <set id='JumpStepTime'>s used to establish the galvo comand update rate. The argument is interpreted as a floating point vector speed in user-units/ second. Example: <set id='JumpSpeed'>**10000**</set>	speed - jump vector speed	user-units/ sec	>0	HW practical limit
JumpStepTime	Sets the update interval to be used in defining the jumping speed when the command <set id='JumpSpeed'> is invoked with a single argument. If *JumpStepTime* is set to 0, then the automatic vector speed calculation algorithm will be invoked per the arguments of the command <set id='VectorSpeedParams'> Example: <set id='JumpStepTime'>**10**</set>	value - JumpSpeed update interval. The default value is 10.	usec	10	HW practical limit
MarkDelay	Set the delay used at the end of a series of marks. Example: <set id='MarkDelay'>**150**</set>	duration - length of time to delay	µsec	0	65535
MarkSpeed (Two argument syntax)	Set the marking 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='MarkSpeed'>**10, 70**</set>	stepTime – the duration between each micro-step. Can be set in 1µsec increments	µsec	10	65535
		stepSize – the distance traveled for each micro-step	bits or user-units/ sec	1 or min user-units/ sec	65535 or max user-units/ sec
MarkSpeed (Single argument syntax)	If invoked with a single argument, the value specified with the command <set id='MarkStepTime'> is used to establish the galvo comand update rate. The argument is interpreted as a floating point vector speed in user-units/ second. Example: <set id='MarkSpeed'>**5000**</set>	speed - mark vector speed	user-units/ sec	>0	HW practical limit
MarkStepTime	Sets the update interval to be used in defining the marking speed when the command <set id='MarkSpeed'> is invoked with a single argument. If *MarkStepTime* is set to 0, then the automatic vector speed calculation algorithm will be invoked per the arguments of the command <set id='VectorSpeedParams'> Example: <set id='MarkStepTime'>**10**</set>	value - MarkSpeed update interval. The default value is 10.	usec	10	HW practical limit
PolyDelay	Set the delay to be used at the junction of two marks. Example: <set id='PolyDelay'>**150**</set>	duration – length of time to delay between two sequential mark vectors	µsec	0	65535

Param Identifier	Description	Arguments	Units	Min	Max
VariJumpDelay	Below a given jumpLengthLimit, the jump delay is linearly scaled down from the JumpDelay value to the minimumDelay as the jump distance approaches 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	user units	1 bit	65535 bits
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	false	true
VectorSpeedParams	Set the limits of the vector speed calculation algorithm used when automatic MarkSpeed and JumpSpeed parameter calculation is desired. These parameters are used when the commands <set id=’MarkSpeed’> or <set id=’JumpSpeed> are invoked with a single argument. Example: <set id=’VectorSpeedParams>**100, 20**</set>	MaxUpdateInterval - the largest update interval that will be tried to optimize the actual vector speed relative to the requested vector speed.	usec	10	HW practical limit
		MaximumErrorPct - that largest permissible error percentage in actual velocity relative to the requested velocity.	%	1	100
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	user units	1 bit	32767 bits

Motion control commands

Coordinate units are controlled buy the <set id=’Units’> command.

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
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 - coordinates of each vertex of a polygon. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent, the value is not changed.	user units	-32768 bits	32767 bits
EnableParkPosition	Permits the “parking” of a scan-head in dual-scanhead systems. Example: <EnableParkPosition>**2**</EnableParkPosition> It is expected that a JumpAbs command is executed prior to this command to move the galvos to the “park” position.	Head selection - heads are selected for the action as follows: 0 - Parking is disabled for both heads 1 - Parking is enabled for head 1 (analog or XY2-extended port) 2 - Parking is enabled for head 2 (normal XY2-100 port) 3 - Both heads are parked.	int	0	3
JumpAbs	Move laser galvos to the absolute position with the laser off Example: <JumpAbs>**-5000; -5000**</JumpAbs>	X, Y and Z [optional] - coordinate of the end of a jump vector. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent, the value is not changed.	user units	-32768 bits	32767 bits
JumpAbsList	Move the laser galvos to the each one of the specified points in succession at the specified update interval with the laser off. Example: <JumpAbsList tick=’**10**’> <Pt> **100; 215; 10**</Pt> <Pt> **110; 240; 30**</Pt> <Pt> **120; 250; 50**</Pt> <Pt> **130; 255; 60**</Pt> </JumpAbsList>	tick - the galvo command update interval	usec	10	65535
		X, Y and Z [optional] - sequence of point coordinates that will be written to the galvos at the rate specified by the “tick” parameter. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent, the value is not changed.	user units	-32768 bits	32767 bits

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
JumpRel	Move the laser galvos to a position relative to the last commanded position with the laser off. Example: <JumpRel>**-25; 50**</JumpRel>	X, Y and Z [optional] - offsets used to calculate a jump vector relative to the lasted commanded position. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent its relative move is set to zero.	user units	-65535 bits	65535 bits
MarkAbs	Move laser galvos to the absolute position with the laser on Example: <MarkAbs>**-5000; 5000; 200**</MarkAbs>	X, Y and Z [optional] - coordinate of the end of a marking vector. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent, the value is not changed.	user units	-32768 bits	32767 bits
MarkAbsList	Move the laser galvos to the each one of the specified points in succession at the specified update interval with the laser on. Example: <MarkAbsList tick=’**10**’> <Pt> **100; 215; 10**</Pt> <Pt> **110; 240; 30**</Pt> <Pt> **120; 250; 50**</Pt> <Pt> **130; 255; 60**</Pt> </MarkAbsList>	tick - the galvo command update interval	usec	10	65535
		X, Y and Z [optional] - sequence of point coordinates that will be written to the galvos at the rate specified by the “tick” parameter. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent, the value is not changed.	user units	-32768 bits	32767 bits
MarkRel	Move the laser galvos to a position relative to the last commanded position with the laser off. Example: <MarkRel>**-355; 500**</MarkRel>	X, Y and Z [optional] - offsets used to calculate a marking vector relative to the lasted commanded position. Values are floating point and are converted into system “bits” units per the <Units> command. If Z is absent its relative move is set to zero.	user units	-65535 bits	65535 bits
WobbleEnable	Enables or disables the wobble function. Wobble parameters should have already been set using the <Set id=’Wobble’> parameter Example: <WobbleEnable>**0**</WobbleEnable>	Wobble enabled state: 0 (disabled) 1 (enabled)	N/A	0	1

Laser control parameters.

Param Identifier	Description	Arguments	Units	Min	Max
LaserEnableDelay *(default set in LaserConfig XML file)*	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’>**10**</set>	delay – the delay from the leading edge of LASERENABLE to the leading edge of LASERON	msec	0	65535
LaserEnableTimeout *(default set in LaserConfig XML file)*	Set the time-out for LASERENABLE to de-assert. A default value can be set in the LaserConfig fixed data as parameter LsrEnaTmo. Example: <set id=’LaserEnableTimeout’>**20**</set>	delay – the time-out from the trailing edge of LASERON to when LASERENABLE is de-asserted	msec	0	65535
LaserFPK *(default set in LaserConfig XML file)*	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>	position – the delay from the leading edge of LASERON to the assertion of the LASERFPK signal	laser timing ticks	-32768	32767
		length – duration of assertion of the LASERFPK signal	laser timing ticks	0	65535
LaserModDelay *(default set in LaserConfig XML file)*	Set the modulation delay of the laser. Example: <set id=’LaserModDelay’>**25**</set>	delay – the delay from the leading edge of LASERON to the output of the first pulse on the LASERMOD1 signal	laser timing ticks	0	65535
LaserOffDelay	Set the delay for turning off the laser when marking Example: <set id=’LaserOffDelay’>**100**</set>	duration – length of time to delay	µsec	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.	µsec	-32768	32767

Param Identifier	Description	Arguments	Units	Min	Max
LaserStandby *(default set in LaserConfig XML file)*	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>	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	laser timing ticks	0	65535
LaserPipelineDelay *(default set in ControlConfig XML file)*	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’>**1500**</set>	delay – the delay that all laser control signals are time shifted relative to micro-vectoring operations.	usec	0	4095
LaserPower	Set the level of the laser power port. The LaserPower port may be the 8-bit digital port or analog port 0 depending on the LaserConfig file setting <LsrPwrPort> Example: <set id=’LaserPower’>**200**</set>	powerValue - value to set the laser power port. If the value is different from the last LaserPower command, then the LaserPowerDelay delay is invoked.	counts	0	255
LaserPowerDelay *(default set in LaserConfig XML file)*	Delay after changing power setting. A default value can be set in the LaserConfig fixed data as parameter LsrPwrDly. Example: <set id=’LaserPowerDelay’>**125**</set>	duration - length of time to delay after setting LaserPower or executing WriteAnalog for port 0	µsec	0	$(2^{32}-1)/50$
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
LaserTiming *(default set in LaserConfig XML file)*	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

Laser control commands

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
LaserEnable	Set the laser active state Example: <LaserEnable>**true**</LaserEnable>	If the laser enabled state is "false" then the special pointer laser mode is activated per the settings of LaserModeConfig. Laser active state: false - disabled true - enabled	Bool	false	true
LaserOn	Turn the laser on for the duration provided. Example: <LaserOn>**1000**</LaserOn>	Duration – length of time to turn the laser on	µsec	1	$(2^{32}-1)/50$
LaserSignalOff	Turns laser off immediately Example: <LaserSignalOff></LaserSignalOff>	N/A			
LaserSignalOn	Turns laser on immediately Example: <LaserSignalOn></LaserSignalOn>	N/A			

External I/O commands

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
WaitForIO	Wait for the digital 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 time-out 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 Association 0 STRTMRK 1-4 USERIN1-4 5 SPAREIN 6-9 INTERLOCK1-4 16-31 Extended DIN bits 0-15	N/A	0	4
		levelPolarity – 0 LowLevel, 1 HighLevel, 2 RisingEdge, 3 FallingEdge	N/A	0	3
		time-out – abort wait if time exceeds the value. If time-out is zero, then wait indefinitely.	µsec	1	85.899 sec
		debounce – debounce the external signal for this period of time	msec	1	65535
WriteAnalog *(defaults are set in LaserConfig XML file)*	Commands the analog output port to a new value. Port 0 is the Laser Power port (A1), and Port 1 is the auxiliary analog output port (A2). A write to port 0 will incur the LaserPowerDelay (see *Laser control parameters.*) Example: <WriteAnalog>**1,344**</WriteAnalog>	portNumber – analog output port identifier	N/A	0	1
		value – new port value	bits	0	4095
WriteDigital *(default for port 102 is set in LaserConfig XML file)*	Commands the digital output port to a new value. Example: <WriteDigital>**3,1**</WriteDigital>	portNumber – port identifier Port Association 0 JOBACTIVE 1-4 USEROUT1-4 5 MRKINPRG 6 ERROR/NREADY 16-31 Extended DOUT bits 0-15 100 System status ports as a group 101 Extended I/O ports as a group 102 8-bit digital power port.	N/A	0	5
		value to send to port. Actual signal polarity is determined by how the optical isolators are connected. Single bit mode (ports 0-31): 0 (unasserted) and 1 (asserted) Group mode (ports 100-102) 0 (unasserted) and 1 (asserted) in bit positions defined as follows: Port 100 bits 0-7 == USEROUT[1-4], MRKINPRG, BUSY, ERROR/NREADY, JOBACTIVE Port 101 bits 0-15 == Ext'd DOUT bits0-15 Port 102 bits 0-7 == Laser digital bits 0-7	UI16	0	65535

Utility commands

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
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 *5.6.2 OnMessageEvent*). Application events should be used sparingly as system performance could be affected if generated at a high rate. Example: <ApplicationEvent>**100,345**</ApplicationEvent>	param1 – First application-specific parameter. Value is returned in OnMessageEvent puiPayloadHigh[31.17]	N/A	0	65536
		param2 – Second application-specific parameter. Value is returned in OnMessageEvent puiPayloadLow[31.17]	N/A	0	2^{32}-1
BeginJob	Generates a BeginJob application event when executed by the marking engine. The <JobDataCntr> parameter in the StatInfoData broadcast packet is re-initialized to zero. BeginJob automatically sets the system BUSY signal. Example: <BeginJob></BeginJob>	N/A			
EndJob	Generates an EndJob application event when executed by the marking engine. The system BUSY signal is automatically cleared. Example: <EndJob></EndJob>	N/A			

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
JobDataCntr	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: <JobDataCntr>**0**</JobDataCntr>	value – counter value (only accepts zero for now)	N/A	0	0
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
LongDelay	Delay all operations for the duration provided Example: <LongDelay>**10000**</LongDelay>	Duration – length of time to sleep	μsec	1	85.899 sec
Set	Set the named parameter to the specified value. Example: <Set id='MarkSpeed'>**10, 2**</set>	The number of and value of the argument(s) is specific to the named parameter.			

Coordinate system transform parameters.

Param Identifier	Description	Arguments	Units	Min	Max
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
FieldOffset	Set an offset to be applied to all vectors at run-time Example: <set id='FieldOffset'>**5000,-1000,100**</set>	XOff, YOff, ZOff(opt.) - Offset in bits to be applied to the vectors at run-time. If Z is not specified, it is set to zero.	user units	-32768 bits	32767 bits
FieldOrientation	Sets the orientation of the marking field relative to the vector coordinate system. This transformation is applied at run-time. Example: <set id='FieldOrientation'>**90**</set>	rotation - specifies the CCW rotation of the marking field in degrees. Allowable settings are: 0, 90, 180, 270.	deg	0	270
Offset	Set an offset to be applied to be applied to the vector set before being passed to the EC1000. Example: <set id='Offset'>**1000,2000,100**</set>	XOff, YOff, ZOff(opt.) - Offset in bits to be applied to the vectors at run-time **only if** TransformEnable was set. If Z is not specified, it is set to zero.	user units	-32768 bits	32767 bits
Transform	Set the values of the coordinate transform matrix to be applied to the vector set before being passed to the EC1000. Example: <set id='Transform'>**1.0, 0.0, 0.0, 1.0**</set>	M00, M01, M10, M11 - 2x2 transform coefficients. These matrix coefficients are applied before the "Offset" values are added.	N/A	-10.0	10.0
TransformEnable	Enables or disables run-time coordinate transformations using the transform selected by the ID argument. The transform information is specified using the Priority data message SetRTJobTransform2D. Example: <Set id='TransformEnable'>**1**</Set>	This does not alter the effect of the Job Offset and Transform parameters. ID == 0, RT transforms are disabled ID == 1, Use transform ID 1 ID == 2, Use transform ID 2	trans-form ID	0	2

Hardware interface configuration parameters

These configuration parameters are set in the configuration files stored on the EC1000 and automatically applied at power-up. They are avaiable here to permit over-riding those settings. .

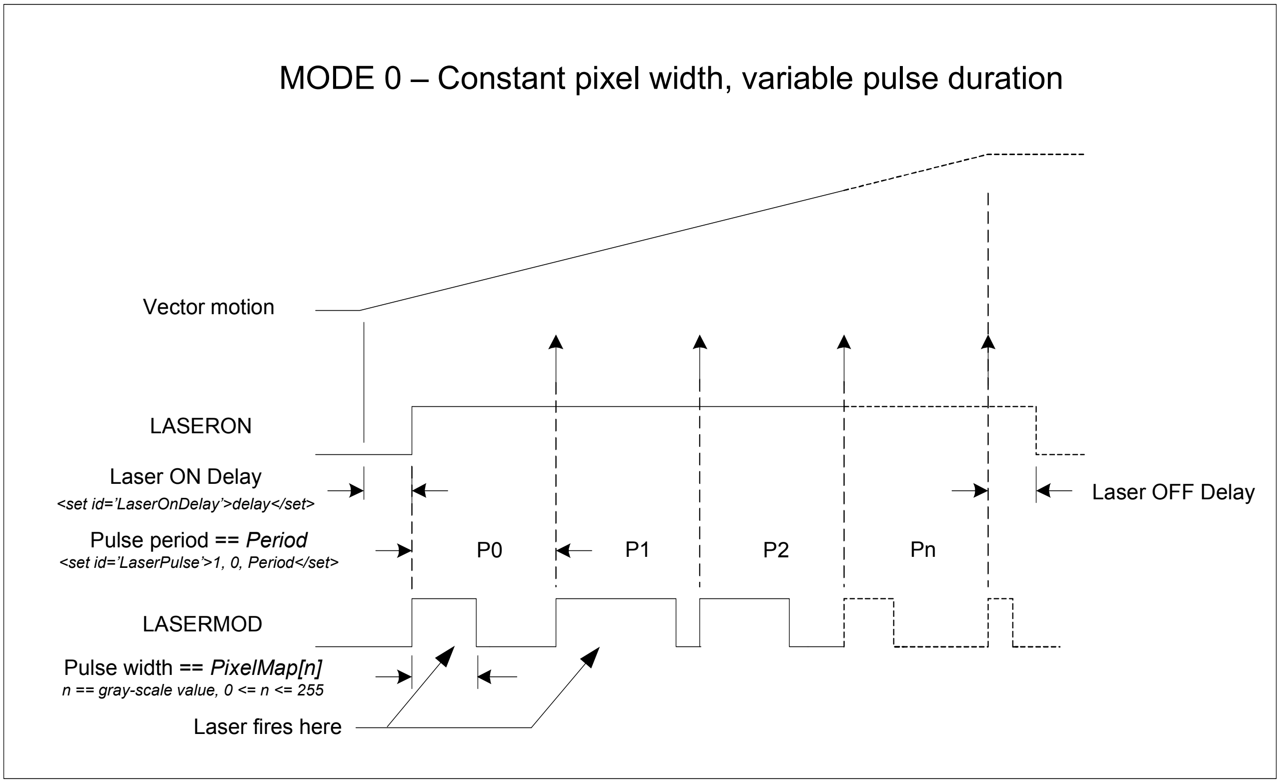
Param Identifier	Description	Arguments	Units	Min	Max
AxisDACRange	Set the analog command output configuration for the laser galvo servo controllers using a bitmask. This is normally set in the ControlConfig XML file but can be over-ridden with this command. Example: <set id='AxisDACRange'>**0x6**</set> // sets the X & Y range to ±10V Z to ±5V	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 single-ended voltage range encoding is as follows: 00 = ±2.5V, 01 = ±5V, 10 = ±10V			

<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
(default set in LaserConfig XML file)</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 individual signals can be set by setting the LaserConfigData elements: LENAHigh, LONHigh, LMOD1High, LMOD2High, LFPKHigh, LON2Cfg</td></tr>
<tr><td colspan="5">
<table>
<tr><th>Name</th><th>Hex Bit 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>
<tr><td>Laser Power Port mode</td><td>0x0200</td><td>Set the mode of the digital laser power port
0=8-bit mode, 1=7-bit mode (LSB used as strobe)</td></tr>
<tr><td>LASERON2 configuration</td><td>0x0800 &
0x0400</td><td>Sets the mode of operation of LASERON2
0 - LASERON2 == !LASERON1
1 - LASERON2 == LASERON1 & !LasersEnabled
2 - LASERON2 == !LasersEnabled
3 - LASERON2 == Asserted all of the time</td></tr>
<tr><td>Laser Power Port</td><td>0x1000</td><td>0=8-bit digital power port, 1=analog output A1</td></tr>
<tr><td>LASERON1 configuration</td><td>0x2000</td><td>0=Gating signal, 1=Modulation signal if 8-bit digital power port bit 7 is also set</td></tr>
</table>
</td></tr>
<tr><td rowspan="2">ServoConfig</td><td rowspan="2">Set the configuration of the X, Y and Z servo control interface.
Example:
<set id=’ServoConfig’>**0x4**</set>
// 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</td></tr>
<tr><td colspan="4">.
<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>

5.3.5 Bit-map raster support

Bit-map raster rendering can be performed in four different modes depending on the level of quality and throughput required. Two “fire-on-the-fly” modes and two “step-and-shoot” modes are supported. These modes are illustrated in the following figures that show the relative galvo motion and laser modulation control.

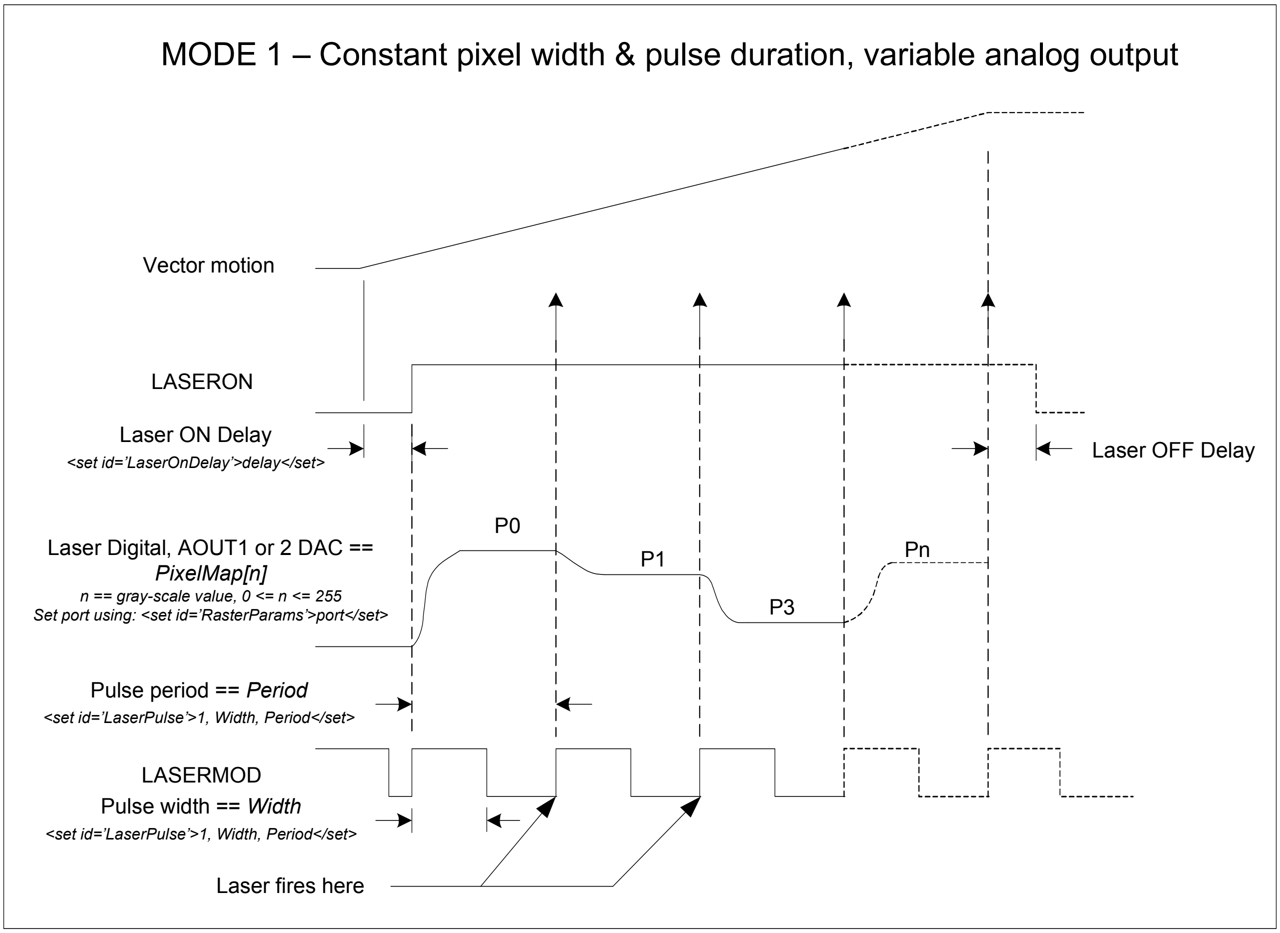
Figure 5: “Fire-on-the-fly”, Mode 0


MODE 0 – Constant pixel width, variable pulse duration
Vector motion
LASERON
Laser ON Delay
<set id=’LaserOnDelay’>delay</set>
Laser OFF Delay
Pulse period == Period
<set id=’LaserPulse’>1, 0, Period</set>
P0
P1
P2
Pn
LASERMOD
Pulse width == PixelMap[n]
n == gray-scale value, 0 <= n <= 255
Laser fires here

Mode 0 raster patterning permits gray-scale imaging when the laser supports variable laser power as a function of how long the laser modulation signal remains on. This is typical of how CO_2 lasers operate. In this illustration, the “high” pulse-width is proportional to an 8-bit gray-scale pixel value. Since the laser fires at a constant rate and the start of the galvo position commands and the start of the lasing process are tightly controlled, the start of each pixel position is accurately placed on the substrate.

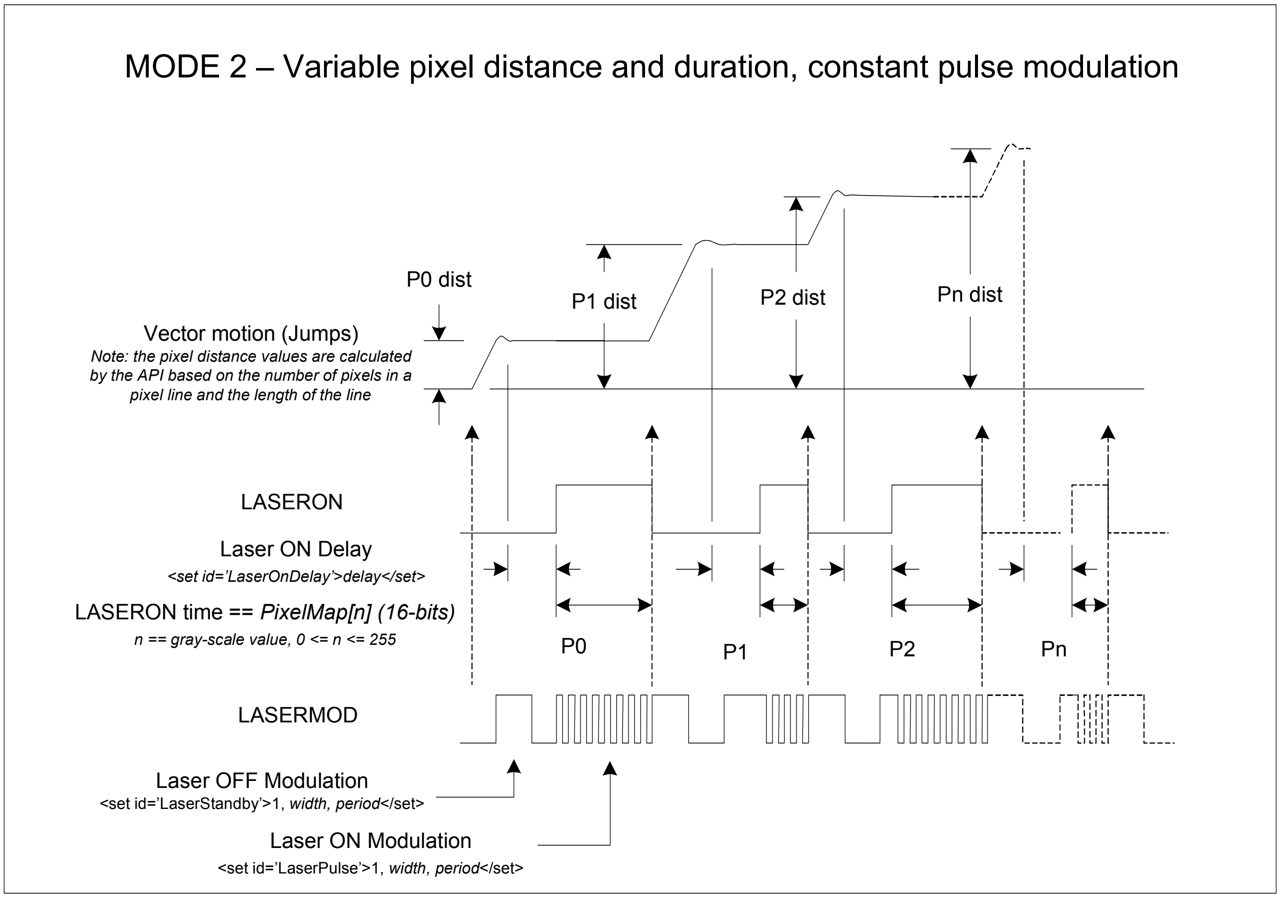
This mode is also very useful in a thresholded or error-diffusion dithering gray-scale approximation approach using Q-Switched lasers. In this case, a low-thresholded or “0” pixel value can cause the pulse-width to be set to zero thus skipping the firing of the laser at that pixel location. Likewise, a high-thresholded or “1” pixel value can cause the laser to fire at that location.

Figure 6: "Fire-on-the-fly", Mode 1


MODE 1 – Constant pixel width & pulse duration, variable analog output
Vector motion
LASERON
Laser ON Delay
<set id='LaserOnDelay'>delay</set>
Laser OFF Delay
Laser Digital, AOUT1 or 2 DAC ==
PixelMap[n]
n == gray-scale value, 0 <= n <= 255
Set port using: <set id='RasterParams'>port</set>
P0
P1
P3
Pn
Pulse period == Period
<set id='LaserPulse'>1, Width, Period</set>
LASERMOD
Pulse width == Width
<set id='LaserPulse'>1, Width, Period</set>
Laser fires here

Mode 1 raster patterning permits gray-scale imaging when the laser supports variable laser pulse power as a function of variable analog or digital laser power control. In this illustration, the laser power control is set proportional to an 8-bit gray-scale value at the beginning of a pixel period and the laser fires at the end of the period on each rising edge of the laser modulation signal. The pulse width of the laser modulation signal is programmable and stays the same for each pixel in the pixel line. Since the laser fires at a constant rate and the start of the galvo position commands and the start of the lasing process are tightly controlled, the pixels positions are accurately placed on the substrate.

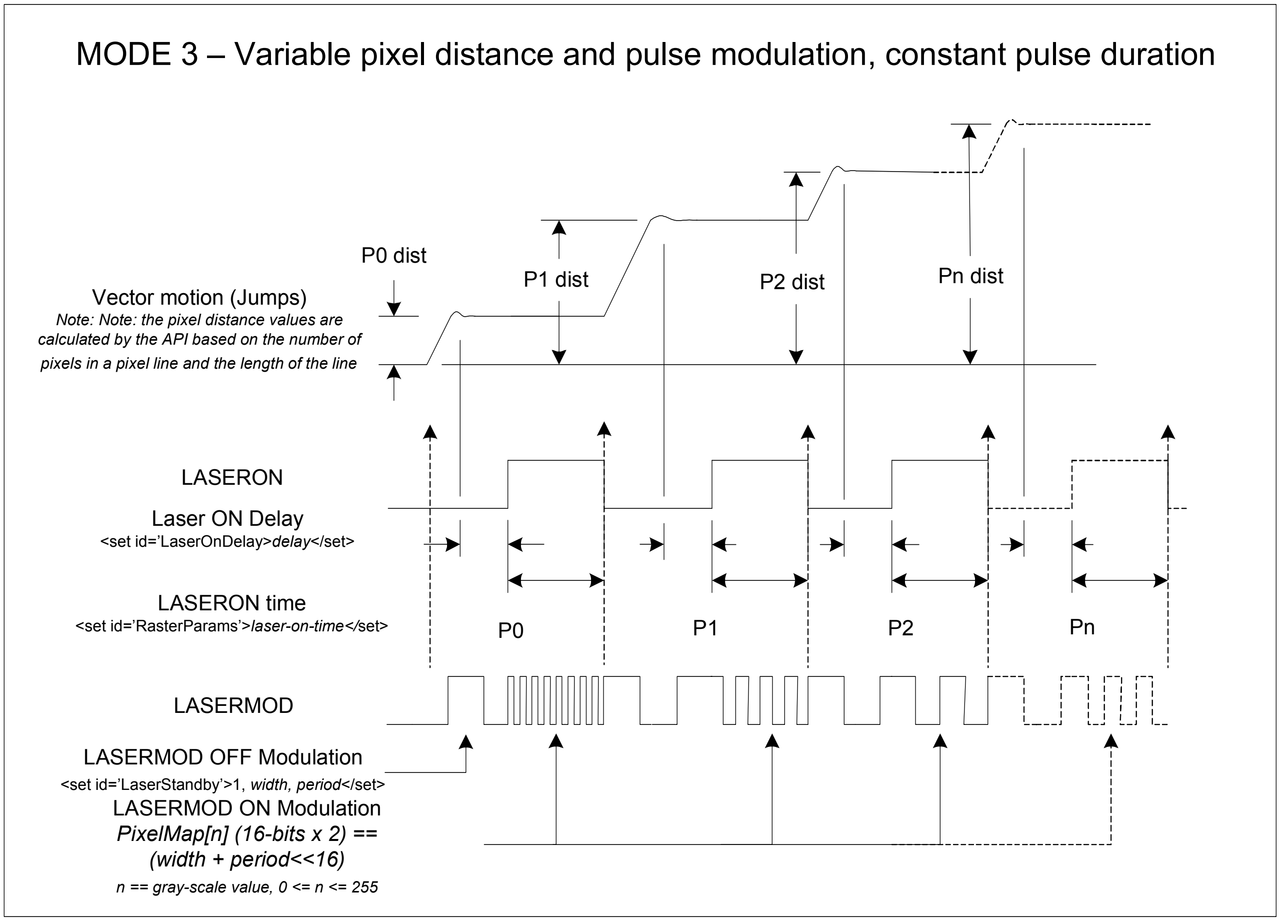
Figure 7: “Jump-and-fire”,Mode 2


MODE 2 – Variable pixel distance and duration, constant pulse modulation
P0 dist
P1 dist
P2 dist
Pn dist
Vector motion (Jumps)
Note: the pixel distance values are calculated by the API based on the number of pixels in a pixel line and the length of the line
LASERON
Laser ON Delay
<set id='LaserOnDelay'>delay</set>
LASERON time == PixelMap[n] (16-bits)
n == gray-scale value, 0 <= n <= 255
P0
P1
P2
Pn
LASERMOD
Laser OFF Modulation
<set id='LaserStandby'>1, width, period</set>
Laser ON Modulation
<set id='LaserPulse'>1, width, period</set>

Mode 2 raster patterning permits gray-scale imaging with most pulsed lasers. Gray scale is achived by controlling the time that a laser fires at a pixel location. The galvos are instructed to jump to each pixel location and the laser fires for a duration that is proportional to the 8-bit grayscale pixel value. In this illustration, the laser modulation characteristics are the same for all pixels on the pixel line, but the laser ontime varies.

Since the galvos jump to each pixel location and stops there before firing, very precise pixel placement is achieved regardless of the scanning direction. Squences of pixels whose value is zero (no laser on time required) are automatically concatenated into a single jump to the next non-zero pixel.

Figure 8: "Jump-and-Fire", Mode 3


MODE 3 – Variable pixel distance and pulse modulation, constant pulse duration
P0 dist
P1 dist
P2 dist
Pn dist
Vector motion (Jumps)
Note: Note: the pixel distance values are calculated by the API based on the number of pixels in a pixel line and the length of the line
LASERON
Laser ON Delay
<set id='LaserOnDelay>delay</set>
LASERON time
<set id='RasterParams'>laser-on-time</set>
P0
P1
P2
Pn
LASERMOD
LASERMOD OFF Modulation
<set id='LaserStandby'>1, width, period</set>
LASERMOD ON Modulation
PixelMap[n] (16-bits x 2) ==
(width + period<<16)
n == gray-scale value, 0 <= n <= 255

Mode 3 raster patterning permits gray-scale imaging with most pulsed lasers on substrates that respond to varying pulse characteristics. Gray scale is achived by controlling the pulse width and period of the laser modulation signal at each pixel location during a fixed laser on time. The galvos are instructed to jump to each pixel location and the laser fires for a fixed duration with a pulse pattern that is proportional to the 8-bit grayscale pixel value.

Since the galvos jump to each pixel location and stops there before firing, very precise pixel placement is achieved regardless of the scanning direction. Squences of pixels whose value is zero (no laser modulation specified) are automatically concatenated into a single jump to the next non-zero pixel.

5.3.6 Bit-map raster commands

Raster operations are defined through the use of the commands defined in *Bit-map raster parameters and commands*. These commands can be freely placed anywhere in a job.

The API supports a pixel mapping table that permits non-linear mapping of 8-bit pixel values to the appropriate laser control values required by the selected mode. This permits a linear range of gray-scale pixel values to scale into a range that is appropriate for the behavior of the laser and materials being used.

Bit-map raster parameters and commands

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
RasterMode	Selects the mode of raster operation. Example: <set id=’Rastermode’>**1**</set>	mode – raster mode per the descriptions above	N/A	0	3
PixelMap	Sets the values of the pixel mapping table. Example: <set id=’PixelMap’>**0,1, 2, ... 255**</set>	table value - 256 entries are used to form a table that is indexed by the actual gray-scale pixel value specified in a <PixelLine> command. The table value indexed by the gray-scale pixel value represents the variable part of laser control system per the selected raster mode. Mode Table value interpretation			
		0 Laser ON pulse width	laser ticks	0	255
		1 Laser power control	bits	0	255
		2 Laser ON time	laser ticks	0	65535
		3 Laser ON pulse width and period (two 16-bit values joined into 32-bits)	laser ticks	0	65535
RasterParams	Sets mode-specific parameters. Example: <set id=’RasterParams’>**1**</set>	Some raster modes require additional parameters to function properly: Mode Parameter			
		1 Pixel output port selection 0 == Analog Port 1 1 == Analog Port 2 2 == 8-bit digital port	N/A	0	2
		3 Laser ON time	ticks	0	65535
RasterLine	Specifies the data and trajectory of a raster line. Example: <RasterLine X=’**10000**’ Y=’**0**’ Z=’**0**’> **25;15;44;100;0;0;33;34, ...**</RasterLine>	A raster line is drawn from the current galvo position to the destination coordinates as specified by the attributes X, Y and Z. In raster modes 2 & 3, the inter-pixel distance is calculated from the number of pixels specified and the length of the vector. The pixel values are ised as indexes into the PixelMap to derive the appropriate hardware values for the selected raster mode.	user units (X, Y & Z)	0	0

5.3.7 Mark-on-the-fly support

Marking on the fly (MOTF) support is provided through the use of several configuration and activation commands. Motion tracking in either the X or Y axis can be configured using a digital quadrature input, or by simulating the motion in situations where an encoder feedback is not avialable but the motion speed is relatively constant.

The MOTF configuration is set using the parameters <MotfCalFactor>, <MotfMode>, and <MotfDirection> defined in the ControlConfig file and additionally changable as part of a job. Run-time control of the MOTF operation is performed through the use of the commands: <MotfEnable>, <MotfWaitForCount>, and <MotfResetJump>. *Figure 9: Mark-on-the-fly process flow* shows the intended use of these commands.

The semantics of the MOTF commands are designed to permit multiple marking sequences within a single job, each of which requires separate frames of data that must precisely spaced in distance. This normally occurs when the required markings exceed the physical limits of the lens field. Wire marking applications are a good example where different information must be marked at precise, but relatively long distances along the length of the wire.

Mark-on-the-fly parameters.

Param Identifier	Description	Arguments	Units	Min	Max
MotfCalFactor *(default set in the ControlConfig XML file)*	Relates laser positioning bits to motion encoder counts. A default value for MotfCalFactor can be set in the ControllerData fixed data structure. **NOTE:** If used in a job, this command must appear after <set id='MotfDirection'> 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
MotfDirection *(default set in the ControlConfig XML file)*	MOTF orientation and direction in degrees. A default value for MotfDirection can be set in the ControllerData fixed data structure. **NOTE:** This command must appear before <set id='MotfCalFactor'> 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
MotfMode *(default set in the ControlConfig XML file)*	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 encoder	N/A	0	1

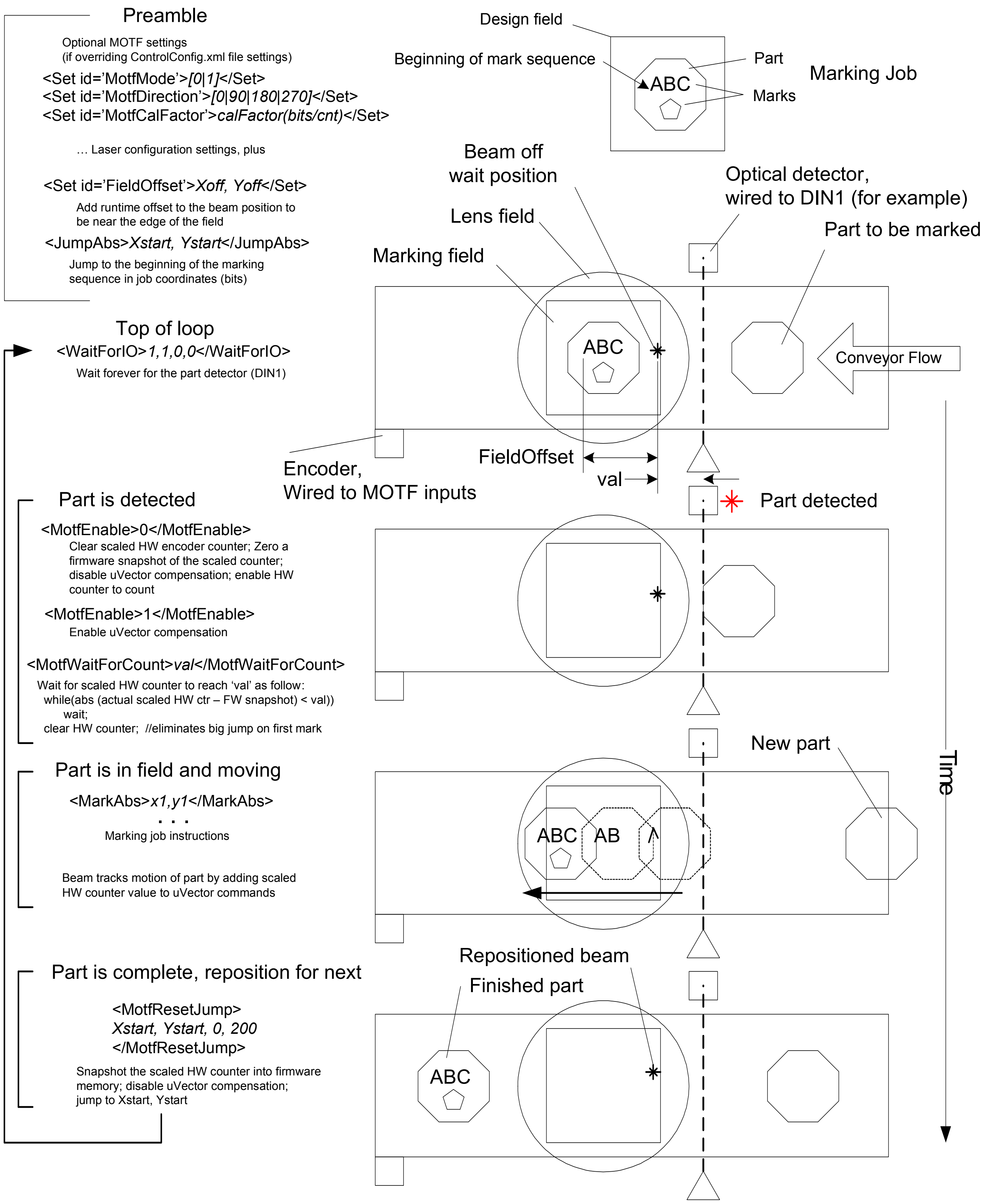
Mark-on-the-fly commands

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
MotfEnable	Enables or disables Mark-on-the-fly (Motf) tracking. Disabling has the following effects: - disables uVector compensation - clears the HW encoder counter - zeros a firmware snapshot of the scaled HW counter - enables the HW encoder counter to count Upon enabling, the scaled Motf encoder counts are added to the uVector values on each Jump and Mark vector. If in simulate mode (see MotfMode), the counter is incremented at a 1Mhz rate. Example: <MotfEnable>0</MotfEnable>	state - 0 disabled, 1 - enabled	N/A	0	1

MotfWaitForCount	Wait for scaled HW encoder counter to reach or exceed a specific value as follow:
	while(abs (actual scaled HW ctr – FW snapshot) < val))
	wait;
	clear HW counter; //eliminates big jump on first mark
	Example:
	<MotfWaitForCount>**24557**</MotfWaitForCount>
		count - scaled encoder count	bits	-2^{31}	$2^{31}-1$

MotfResetJump	When encountered the following actions occur: - a snapshot of the scaled HW counter is taken and saved into firmware memory - uVector compensation is disabled - a jump to Xstart, Ystart is done Example: <MotfResetJump>**-23000,400,0,200**</MotfResetJump>	X, Y, Z - coordinate	bits	-32768	32767
		JumpDelay	usec	0	65535

Figure 9: Mark-on-the-fly process flow



Preamble

Optional MOTF settings
(if overriding ControlConfig.xml file settings)

```
<Set id='MotfMode'>[0|1]</Set>
<Set id='MotfDirection'>[0|90|180|270]</Set>
<Set id='MotfCalFactor'>calFactor(bits/cnt)</Set>
```

… Laser configuration settings, plus

```
<Set id='FieldOffset'>Xoff, Yoff</Set>
```

Add runtime offset to the beam position to be near the edge of the field

```
<JumpAbs>Xstart, Ystart</JumpAbs>
```

Jump to the beginning of the marking sequence in job coordinates (bits)

Top of loop

```
<WaitForIO>1,1,0,0</WaitForIO>
```

Wait forever for the part detector (DIN1)

Part is detected

```
<MotfEnable>0</MotfEnable>
```

Clear scaled HW encoder counter; Zero a firmware snapshot of the scaled counter; disable uVector compensation; enable HW counter to count

```
<MotfEnable>1</MotfEnable>
```

Enable uVector compensation

```
<MotfWaitForCount>val</MotfWaitForCount>
```

Wait for scaled HW counter to reach 'val' as follow:

```
  while(abs (actual scaled HW ctr – FW snapshot) < val))
      wait;
  clear HW counter;  //eliminates big jump on first mark
```

Part is in field and moving

```
<MarkAbs>x1,y1</MarkAbs>
. . .
```

Marking job instructions

Beam tracks motion of part by adding scaled HW counter value to uVector commands

Part is complete, reposition for next

```
<MotfResetJump>
Xstart, Ystart, 0, 200
</MotfResetJump>
```

Snapshot the scaled HW counter into firmware memory; disable uVector compensation; jump to Xstart, Ystart

Instructions making up the MOTF loop can be sent to the EC1000 in advance of them being required as long as the job data does not vary. Synchronization with the external detectors is handled completely in the EC1000.

5.3.8 Structured job data

Any job data defined above, from single statement to a lengthy sequence of statements, can be passed to the EC1000 for immediate execution via the sendStreamingData() method. Data sent like this is executed once and then discarded. If a repetitive marking pattern is desired, an application could repeatedly send the job data with a sequence of calls to sendStreamingData(). Alternatively, jobs can be structured into groups of related statements called segments and these segments sent to the EC1000 as a named entity for deferred execution. Many segment definitions may be sent to the EC1000 in this manner. A separate sequence list can then be used to dictate the execution order of the segments, how many times to iterate each segment, and how many times to iterate the sequence as a whole.

An entire job made up of multiple segments, and potentially multiple sequences can be sent in a single sendStreamingData() call. The same XML that makes up this job can be passed to the saveJobData() methodfor storage on the EC1000 and later access in stand-alone operational mode.

One or more segment definitions may be also specified and saved as a library for later reference and use within a sequence specification. This greatly reduces the amount of data moving through the system when commonly used graphical entities such as pre-rendered character sets are required at run-time.

Structured job commands

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
Segment	Valid only in a job <Data> definition. Defines a group of related instructions. Any job action command or parameter statement is valid inside of a Segment. Multiple Segments can be defined inside of a call to sendStreamData() Example: <Segment id=’LaserCfg’ iterations=’1’ deferred=’true’> <set id=’LaserPulse’>**1,50,100**</set> <LaserPower>**200**</LaserPower> </Segment> <Segment id=’Vectors’ iterations=’2’ deferred=’true’> <JumpAbs> -1000, 1000,0</JumpAbs> <MarkAbs> 1000, 1000,0</JumpAbs> <MarkAbs> 1000, -1000,0</JumpAbs> <MarkAbs> -1000, -1000,0</JumpAbs> <MarkAbs> -1000, 1000,0</JumpAbs> </Segment>	id - A name assigned to this segment	Char	1	128
		iterations - number of times this segment is to be iterated. Default is 1 if not specified.	N/A	1	65535
		deferred - execute immediately (false) or save for reference by a Sequence (true). Default is false if not specified.	Bool	false	true
Sequence	Valid only in a job <Data> definition. Defines the sequence of execution of previously defined job segments. Example: <Sequence iterations=’3’> <RunSegment>LaserCfg</RunSegment> <RunSegment>Vectors, 5</RunSegment> </Sequence>	iterations - number of times to execute this segment list. A value of 0 means to execute this sequence continuously.	N/A	0	65535
RunSegment	Valid only inside of a <Sequence> definition. Causes the previously loaded and “deferred” named segment to be executed. Example: <RunSegment>Vectors, 5</RunSegment>	segment name - identifier of a previously loaded <Segment>	Char	1	128
		iteration - number of times to iterate the named <Segment>. If the previously loaded <Segment> had an iterations attribute specified, then the two iterations values are multiplied the result is the final iteration count. If not specified, the default is 1.	N/A	1	65535
DeleteSegment	Valid only inside of a <Sequence> definition. The named previously loaded and “Deferred” segment is discarded with all used memory returned to the main memory pool. Example: <DeleteSegment>LaserCfg</DeleteSegment>	segment name - identifier of a previously loaded <Segment>	Char	1	128
DeleteAllSegments	Valid only inside of a <Sequence> definition. All previously loaded and “Deferred” segments are discarded with all used memory returned to the main memory pool. Example: <DeleteAllSegments></DeleteAllSegments>	N/A			
DisableSegment	Valid only inside of a <Sequence> definition. The named previously loaded and “Deferred” segment is marked as “disabled” which causes it to be skipped when encountered within a subsequent sequence list. Example: <DisableSegment>LaserCfg</DisableSegment>	segment name - identifier of a previously loaded <Segment>	Char	1	128

Command Tag	Action Description/Example XML Syntax	Parameters	Units	Min	Max
EnableSegment	Valid only inside of a <Sequence> definition. The named previously loaded and "Deferred" segment is marked as "enabled" which causes it to be executed when encountered within a subsequent sequence list. Example: <EnableSegment>LaserCfg</EnableSegment>	segment name - identifier of a previously loaded <Segment>	Char	1	128
UsingFile	Valid only in a job <Data> definition. This specifies the name of a previously saved set of <Segment> definitions for use in a following <Sequence> definition. Example: <UsingFile>LaserSettings</UsingFile>	segment file name - identifier of a previously saved set of <Segment> definitions. These definitions would have been saved to the EC1000 using the API method saveJobData()	Char	1	128

Note: Do not mix deferred and non-deferred segments in a single XML job packet.

Structured job example

XML Job Statement	Meaning	API Action
<Data type=’JobData’ rev=’2.0’>	Define a job data packet	Prepare a job packet
<Segment id=’Preamble’ iterations=’1’ deferred=’TRUE’>	Define a deferred execution segment	Compile and mark for on-the-board in-memory staging; append to output buffer
<BeginJob></BeginJob>	Assert BUSY signal and generate an event	
<set id=’ActiveCorrectionTable’>1</set>	Select the marking laser correction table	
<set id=’EnableLaser’>TRUE</set>	Enable the laser for marking	
</Segment>	Delimit the segment	
<Segment id=’Alignment’ deferred=’TRUE’>	Define an immediate execution segment	Compile and mark for on-the-board in-memory staging; append to output buffer
<set id=’FieldOffset’>0.000000,0.000000,0.000000</set>	Introduce a field offset	
<set id=’Transform’>1.000000,0.000000,0.000000,1.000000</set>	Set a Unity transform	
</Segment>	Delimit the segment	
<Segment id=’Params:Default’ deferred=’TRUE’>	Define a deferred execution segment	Compile and mark for on-the-board in-memory staging; append to output buffer
<set id=’LaserPower’>50</set>	Set the laser parameters	
<set id=’LaserEnableDelay’>15</set>		
<set id=’LaserEnableTimeout’>15</set>		
<set id=’LaserOnDelay’>0</set>		
<set id=’LaserOffDelay’>50</set>		
<set id=’LaserPipelineDelay’>100</set>		
<set id=’LaserPulse’>1,5,10</set>		
<set id=’MarkSpeed’>10,10</set>	Set the galvo speeds	
<set id=’JumpSpeed’>10,10</set>		
</Segment>	Delimit the segment	
<Segment id=’Vectors:Pentagon.plt’ iterations=’1’ deferred=’TRUE’>	Define a deferred execution segment	Compile and mark for on-the-board in-memory staging; append to output buffer
<set id=’JumpDelay’>100</set>	Set the delays. These must be kept with the vector definitions.	
<set id=’MarkDelay’>100</set>		
<set id=’PolyDelay’>50</set>		
<set id=’VariPolyDelayFlag’>FALSE</set>		
<JumpAbs>-10000,10000,0</JumpAbs>	Perform marking operations	
<MarkAbs>0,20000,0</MarkAbs>		
<MarkAbs>10000,10000,0</MarkAbs>		
<MarkAbs>7500,-10000,0</MarkAbs>		
<MarkAbs>-7500,-10000,0</MarkAbs>		
<MarkAbs>-10000,10000,0</MarkAbs>		
</Segment>	Delimit the segment	
<Segment id=’Postamble’ iterations=’1’ deferred=’TRUE’>	Define a deferred execution segment	Compile and mark for on-the-board in-memory staging; append to output buffer
<set id=’EnableLaser’>FALSE</set>	Enables the pointer laser	
<set id=’ActiveCorrectionTable’>2</set>	Select the pointer laser correction table	
<EndJob></EndJob>	De-assert BUSY signal and generate an event	
</Segment>	Delimit the segment	
<Sequence iterations=’1’>	Define a sequence to be iterated 1 time	Compile and mark for on-the-board in-memory staging; append to output buffer
<RunSegment>Preamble</RunSegment>	Execute the preamble segment	
<RunSegment>Alignment</RunSegment>	Execute the alignment segment	
<RunSegment>Params:Default</RunSegment>	Execute the params segment	
</Sequence>	End the sequence	
<Sequence iterations=’10’>	Define a sequence to be iterate 10 times	Compile and mark for on-the-board in-memory staging; append to output buffer
<RunSegment>Vectors:Pentagon.plt</RunSegment>	Execute the marking vectors	
</Sequence>	End the sequence	
<Sequence iterations=’1’>	Define a sequence to be iterated 1 time	Compile and mark for on-the-board in-memory staging; append to output buffer
<RunSegment>Postamble</RunSegment>	Execute the postamble segment	
</Sequence>	End the sequence	
</Data>	End the job packet	API action: Send output buffer to board Controller actions: Deferred segments are staged in memory on the controller. Each sequence is executed in order.

The job could have been organized differently and achive the same effect, e.g. the immediate segments could have been combined into one segment. The partitioning, however, illustrates how a job can be organized and partitioned into related groups of job commands. This partitioning does not add any run-time overhead.

5.4 Marking job control and administration

Once created in XML format, job data can be sent to an EC1000 after a session has been created using the Session.loginSession() method. The primary interface for sending job data to the EC1000 is the Session.sendStreamData() method.

5.4.1 sendStreamData

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. The special case of zero means to wait // an infinite duration.
	) As Unsigned Long	// Error codes are defined in section *5.7 Session API Error Codes*
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 Definition 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.	
See also		

If a syntax error is detected in the XML job data, an OnData event is generated to relate back to the application the nature of the error. See section for more information.

5.4.2 sendStreamData2

Command	ILecSession.sendStreamData2	
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. The special case of zero means to wait // an infinite duration.
	ByVal bWaitForAck As Boolean	// If set to TRUE, the function does not return until a reception acknowlegement // is received from the EC1000. Otherwise data packets are queued for execution.
	ByRef puiExecutionTime	// Returns an estimated execution time in ms for streaming style packets
	) As Unsigned Long	// Error codes are defined in section *5.7 Session API Error Codes*
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 Definition 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.	
See also		

If a syntax error is detected in the XML job data, an OnData event is generated to relate back to the application the nature of the error. See section for more information.

5.4.3 saveJobData

When a job has been constructed and tested using on-line worktation facilities, it can be sent to the EC1000 for storage on resident Flash memory or on attached USB Flash storage drives 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	// Error codes are defined in section *5.7 Session API Error Codes*

Explanation	Job data is compiled and stored on the target EC1000 Flash file system for use in Local Mode
See also	LecSession.manageJobData(), ILecSession.requestJobNameList()

5.4.4 manageJobData

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 // Error codes are defined in section *5.7 Session API Error Codes*
Explanation	Jobs already stored on the EC1000 can be renamed or deleted.
See also	ILecSession.saveJobData(), ILecSession.requestJobNameList()

5.4.5 requestJobNameList

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 // Error codes are defined in section *5.7 Session API Error Codes*
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>
See also	ILecSession.saveJobData(), ILecSession.manageJobData()

5.5 Priority communication

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. This mechanism is used to querey and EC1000 for status information on-demand in cases where the cycle-time of broadcast packets is insufficient.

5.5.1 sendPriorityData

.

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 // Error codes are defined in section *5.7 Session API Error Codes*
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 "GetRegisters".
See also	

Priority Messages

Priority messages are sent using the Session.sendPriorityData() method. The messages are expressed in XML format as described in *Table 6: Priority message definitions*.

Table 6: Priority message definitions

<table>
<tr><th>Message</th><th>XML Example Syntax</th><th>Description</th></tr>
<tr><td>Abort</td><td>&lt;Data type='ServiceData' rev='1.1'>
<Msg id='Abort' reason='Terminate'/>
&lt;/Data&gt;</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 (<i>5.6.2 OnMessageEvent</i>) 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>&lt;Data type='ServiceData'>
<Msg id='Restart'/>
&lt;/Data&gt;</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>&lt;Data type='ServiceData'>
<Msg id='Suspend'/>
&lt;/Data&gt;</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 including poly-vector mark.</td></tr>
<tr><td>Resume</td><td>&lt;Data type='ServiceData'>
<Msg id='Resume'/>
&lt;/Data&gt;</td><td>Job execution is permitted to continue.</td></tr>
<tr><td>GetRegisters</td><td>&lt;Data type='ServiceData'>
<Msg id='GetRegisters'/>
&lt;/Data&gt;</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 <i>5.6.3 OnDataEvent</i>) message.</td></tr>
<tr><td>SetInterlockEnable</td><td>&lt;Data type='ServiceData'>
<Msg id='SetInterlockEnable' config='<b>0x14</b>'/&gt;
&lt;/Data&gt;
<b>or</b>
<Data type='ServiceData'>
<Msg id='SetInterlockEnable'&gt;<b>0x14</b>&lt;/Msg&gt;
</Data&gt;</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>
<tr><td>SetOffset</td><td>&lt;Data type='ServiceData'>
<Msg id='SetOffset'&gt;<b>200;300;100</b>&lt;/Msg&gt;
</Data&gt;</td><td>Sets the run-time X, Y, and Z offset to be applied to the vectors if the TransformEnable job command had been set to the enabled state. Otherwise, this message has no effect. The Z offset is optional and if not present it is not changed. Units are defined by the <set id='Units'> command</td></tr>
</table>

Message	XML Example Syntax	Description

SetTransform	<Data type=’ServiceData’>
	  <Msg id=’SetTransform’>
	    0.707, -0.707, 0.707, 0.707</Msg>
	</Data>
		Sets the run-time coordinate transform {M00, M01, M10, M11} to be applied to the vectors if the TransformEnable job command had been set to the enabled state. Otherwise, this message has no effect.

StopCurrentSequence	<Data type=’ServiceData’> <Msg id=’StopCurrentSequence’></Msg> </Data>	If a sequence is continuously executing, it is stopped at the end of the current iteration. Other sequences that are queued are run in order.
StopAllSequences	<Data type=’ServiceData’> <Msg id=’StopAllSequences’></Msg> </Data>	If a sequence is continuously executing, it is stopped at the end of the current iteration. Any other sequences that are queued are not run.
Flush	<Data type=’ServiceData’> <Msg id=’Flush’></Msg> </Data>	All queued job data is flushed. Data that has reached the EC1000 marking engine is allowed to complete execution.
SetRTJobTransform2D	<Data type=’ServiceData’> <Msg id=’SetRTJobTransform2D’>**1, 25.0, 0.0, 5.0** </Msg> </Data>	Sets the run-time coordinate transform *{Angle, Xoff, Yoff}* to be applied to the vectors if the TransformEnable job command had been set to the the *id* value. Otherwise, this message has no effect. Arguments: *id* - 1 or 2 to select between two separate transform data sets *Angle* - Angle in degrees to rotate *Xoff*,*Yoff* - X and Y offset values to apply

5.6 Asynchronous communication

The EC1000 API uses programmed events to communicate asynchronous data back to an application. Events are divided into three types: Connect, Message and Data. Connect events are generated on major system state changes during login and logout operations Message events are generated during normal execution of a job. They may be application specified to occur at specific points during job execution, or may be generated by the system to signal an exception condition. Data events are created in response to specific application requests for data from the system or errors generated by the client API or EC1000 server firmware. This permits a non-blocking request/response code structure that is more efficient for data requests that take time to resolve.

5.6.1 OnConnectEvent

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	

5.6.2 OnMessageEvent

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,

Command	ILecSession.OnMessageEvent
Purpose	Application and exception events returned from the EC device session.
Usage	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
Explanation	

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 such as the response to an abort message, servo error detection, etc. The data that classifies the event are passed back as two 32-bit payloads from the controller.

puiPayloadHigh is encoded in two 16-bit entities:

- puiPayloadHigh[15..0] contains the event type
- puiPayloadHigh[31..16 contains event-type specific data

The following event types are supported (puiPayloadHigh[15..0]):

Event Type	Value
Begin Job	0x0041 (65)
End Job	0x0042 (66)
Application Event	0x5040 (20544)

If the event type is an Application Event, then puiPayloadHigh[31..16] define the application message type (see *Table 7: .OnMessageEvent message types*). 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.

The MarkProgress and CycleProgress events, are automatically generated by the EC1000 as a job is executed provided that <JobMarker> commands are inserted into the XML jobstream. The percentage complete values are calculated using information from the <JobMarker> instructions.

Table 7: .OnMessageEvent message types

Message Type	**Value**	**Description**	puiPayloadLow
BeginObject	0x0002 (2)	Beginning of a marking object	Object Identifier
EndObject	0x0003 (3)	End of a marking object	Object Identifier
BeginTask	0x0004 (4)	Beginning of a task object	Task Identifier
EndTask	0x0005 (5)	End of a task object	Task Identifier
MarkProgress	0x0006 (6)	Progress for a marking object	Percent Complete
CycleProgress	0x0007 (7)	Progress for a job	Percent Complete
Reserved	0x0008-0x000E (8-14)	Reserved for future CTI use	Reserved
FixDataProcessed	0x000F(15)	Fixed data update complete	0
Reserved	0x0010-0x00FF (16-255)	Reserved for future CTI use	Reserved
User defined	0x0100-0x1FFF (256-8191)	User defined	User defined
Exception	0x2000-0xAFFF (8192-45055)	Exception messages (see below)	Exception specific
AbortException	0x2329 (9001)	An Abort was processed	0
InterlockException	0x233D (9021)	Interlock was tripped	Interlock bit mask
CmdProcException	0xA678 (42616)	Command processing exception	Exception type code
Reserved	0xB000-0xFFFF (45056-65535)	Reserved for future CTI use	Reserved

A CmdProcException is further refined by the puiPayloadLow value as defined in *Table 8: Exception message event codes*.

Table 8: Exception message event codes

PayloadLow **Code**	**Description**		PayloadLow **Code**	**Description**
0x232A (9002)	Command FIFO empty time-out		0x2337 (9015)	SetMotfDirection bad argument
0x232B (9003)	Event ISR time-out		0x2338 (9016)	EnableMotf bad argument
0x232C (9004)	Bad opcode		0x2339 (9017)	SetLaserPulseWidthPct bad argument
0x232D (9005)	Internal firmware bug		0x233A (9018)	SetLaserPulsePeriodPct bad argument
0x232E (9006)	WriteDigital bad argument		0x233B (9019)	SetFieldOrientation bad argument
0x232F (9007)	SetLaserPower bad argument		0x233C (9020)	Servo fault detected
0x2330 (9008)	SetCorrectionTable bad argument		0x233E (9022)	WriteAnalog bad argument
0x2331 (9009)	SetLaserPulse bad argument		0x2392 (9106)	OuputDrawList bad argument
0x2332 (9010)	WaitForIO bad argument		0x2393 (9107)	Jump/MarkListAbs bad argument
0x2333 (9011)	WaitForIO command time-out			
0x2334 (9012)	SetLaserStandby bad argument			
0x2335 (9013)	Communications time-out to the CPLD			
0x2336 (9014)	Time-out waiting for the laser to go active			

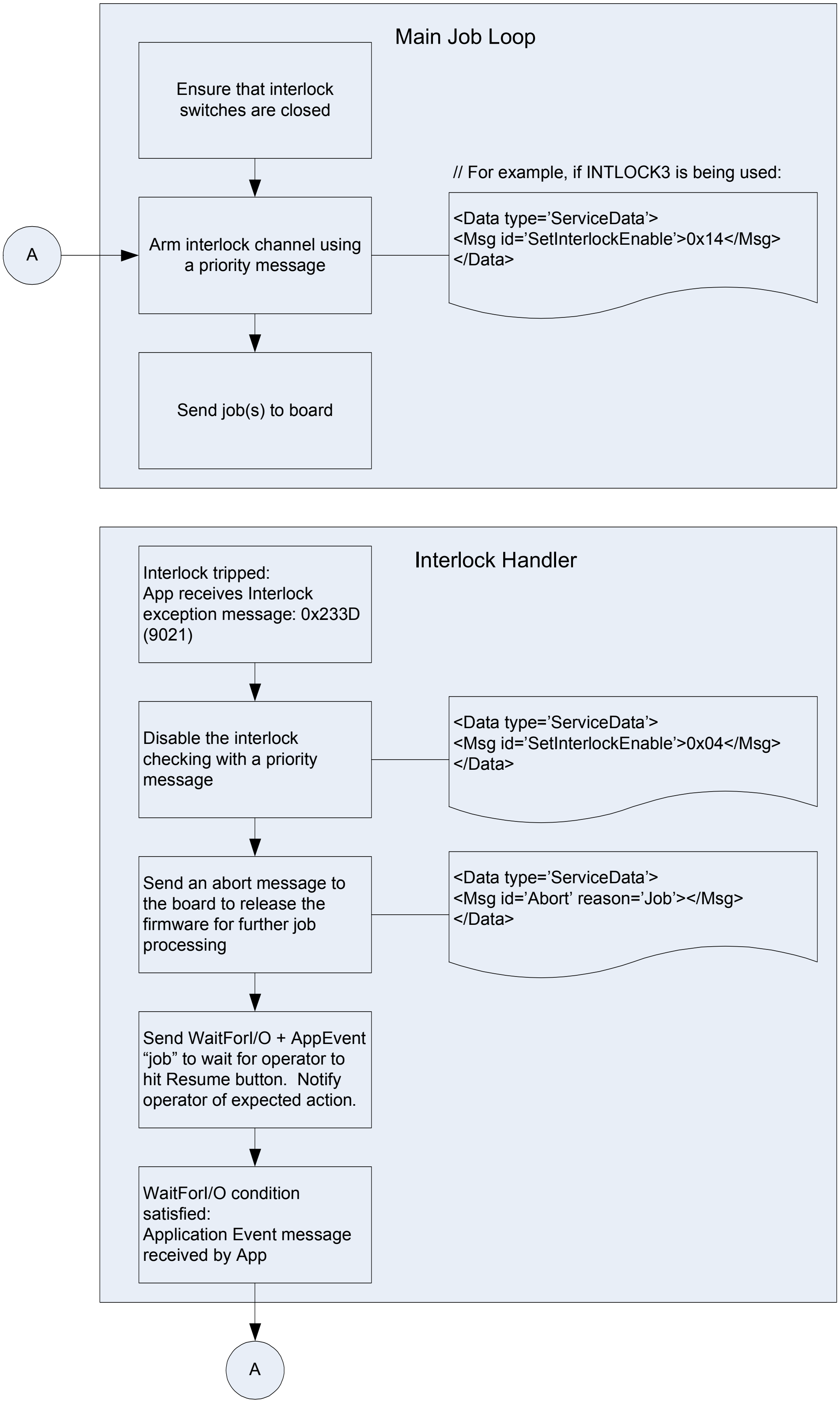
Special notes on interlocks and handling exceptions in general

Exceptions generally indicate that something bad has happened and that marking operations should be terminated as quickly as possible. This is especially important when high-power lasers are involved. The EC1000 provides for fast controlled shut-down of laser operations whenever an exception is detected by the hardware. The interlock system is a safety feature where breaks in the interlock connectivity can be conditioned to shut down the laser and galvo motions, and generate an exception event to the host application to notify it that the break occured.

When a conditioned interlock trips or any other hardware-detectable exception condition occurs, the marking engine controller immediately stops processing the vector stream, turns off the laser, and stops the galvo motion. It then sends an excception event message the host application and enters a state where it will not execute any more instructions until a Priority Abort:Job message is received. The Abort message reinitializes the marking engine and prepares it for a new job. If an exception occurs, the job cannot be retarted from where is left off.

The following figure illustrates a sample protocol for handling and interlock break where the primary user interaction is through a mechanical switch to start/resume a job.

Figure 10: Sample interlock processing protocol



Ensure that interlock switches are closed

Arm interlock channel using a priority message

// For example, if INTLOCK3 is being used:

```
<Data type='ServiceData'>
<Msg id='SetInterlockEnable'>0x14</Msg>
</Data>
```

Send job(s) to board

Interlock tripped:
App receives Interlock exception message: 0x233D (9021)

Disable the interlock checking with a priority message

```
<Data type='ServiceData'>
<Msg id='SetInterlockEnable'>0x04</Msg>
</Data>
```

Send an abort message to the board to release the firmware for further job processing

```
<Data type='ServiceData'>
<Msg id='Abort' reason='Job'></Msg>
</Data>
```

Send WaitForI/O + AppEvent "job" to wait for operator to hit Resume button. Notify operator of expected action.

WaitForI/O condition satisfied:
Application Event message received by App

5.6.3 OnDataEvent

OnDataEvent is used to passes error details or requeted data back to an application. 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.

The system will also generate a Data Event if there is a Job data syntax error. In this case, the suspect fragment of XML is returned as the event data along with an explanatory error message.

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 - Client Errors // 2 - Server Errors // 3 - Registers Data // 4... Reserved ByVal puiErrorCode as Unsigned Long, // Error code returned from the EC1000. No error == 0. ByVal pstrData as String // The data sent by the API. This data can originate from: // - the API in the case of an XML command parsing error // - from the server in the case where a SW exception is detected // - from the EC1000 HW in the case where register data is requested. // The string supplied contains an XML representation of the data.) As Unsigned Long

GetRegisters Priority Message OnDataEvent Response

Explanation	Data is returned asynchronous from the request.
	Register Data is returned as follow:
	<Data type='HardwareState' rev='2.0'>
	<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] == USERIN[0], STRTMRK
	// bits[9..6] == INTERLOCK[4..1]
	// bits[13..10] == USEROUT4..1]
	// bits[17..14] == JOBACTIVE, ERROR/NREADY, 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] == JOBACTIVE, ERROR/NREADY, BUSY, MRKINPRG,
	// STRTMRK
	<DIO.Interlock>**0xF**</DIO.Interlock> // bits[3..0] == INTLOCK[4..1]
	<XVectCmd>**-500**</XVectCmd>
	<YVectCmd>**-500**</YVectCmd>
	<ZVectCmd>**0**</ZVectCmd>
	<AuxIO_Ana1>**0x20**</AuxIO_Ana1> Optional auxiliary I/O module with analog sub-option
	<AuxIO_Ana2>**0x0**</AuxIO_Ana2> Optional auxiliary I/O module with analog sub-option
	<AuxIO_DIn>**0x8FFC**</AuxIO_DIn> Optional auxiliary I/O module
	<AuxIO_DOut>**0x0200**</AuxIO_DOut> Optional auxiliary I/O module
	</Data>

5.7 Session API Error Codes

Errors returned by the Session API are defined in *Table 9: Session API Error Codes*. The error descriptions can be accessed through the use of the method GetErrorCodeDescription(int ErrorCode).

Table 9: Session API Error Codes

Error Name	Code	Description
Success	0	Operation successful
Error_AccessDenied	1	TCP/IP networking access was denied
Error_Communications	2	TCP/IP network communications error occured
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

6 Remote Control API

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*.

6.1 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.

6.2 RS232 Interface

Remote control of the EC1000 can also be established by any host computer that supports RS232 serial communications. Communication is established by opening a COM port connection on the local computer that is connected to the EC1000. The EC1000 COM port that is used for the protocol is controlled by settings in the AdminConfig file. See the section on *Administration configuration* , APIPort, for additional details.

If a single new-line character is sent to the remote control port, the EC1000 transmits a "Welcome banner". This string can be used to verify that communication has been established.

6.3 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 can be either text strings or numeric identifiers 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.

RemoteAdminstrator.exe is a sample program that uses the Remote API to access the EC1000. It is located in C:\Program Files\CTI\Client.

6.3.1 Control and Communications Commands

Abort (1)	
Purpose:	Stops the execution of a job
Implementation:	"Abort" **or** "1"
Parameters:	
Returns:	"0" (command acknowledge)
Comments:	Immediately stops the execution of a running job and sets the *JobRunning* status
See also:	

TakeHostControl (2)	
Purpose:	Requests exclusive control of the EC1000
Implementation:	"TakeHostControl" **or** "2"
Parameters	
Returns:	"0" (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*

ReleaseHostControl (3)	
Purpose:	Releases exclusive control of the EC1000 to the LANStream host interface
Implementation:	"ReleaseHostControl" **or** "3"
Parameters:	
Returns:	"0" (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*

GetHostControlStatus (4)	
Purpose:	Returns the current EC1000 control status of this remote control session
Implementation:	"GetHostControlStatus" **or** "4"
Parameters	
Returns:	"4" (HOST_IN_CONTROL: control has been granted to this session) "5" (HOST_NOT_IN_CONTROL: this session is not in exclusive control of the EC1000)
Comments:	
See also:	*TakeHostControl, ReleaseHostControl*

GetHostInControl (5)	
Purpose:	Returns the current host interface that has exclusive control of the EC1000
Implementation:	"GetHostInControl" **or** "5"
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*

EnableBroadcasting (6)	
Purpose:	Enables/Disables the broadcast function of the EC1000
Implementation:	"EnableBroadcasting, <enable-state>" **or** "6, <enable-state>"
Parameters:	<enable-state> (0 == disable, 1 == enable)
Returns:	"0" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command.
See also:	

LoadHardwareDefaults (7)	
Purpose:	Sets the current operating parameters of the EC1000 to their default values
Implementation:	"LoadHardwareDefaults" **or** "7"
Parameters:	
Returns:	"0" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command.
See also:	

HardwareReset (8)	
Purpose:	Forces a hard reset of the EC1000
Implementation:	"HardwareReset" **or** "8"
Parameters:	
Returns:	"0" (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:	

GetRemoteIP (9)	
Purpose:	Returns the IP address of the LAN stream host that has control of the EC1000
Implementation:	"GetRemoteIP" **or** "9"
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:	

GetKFactor (10)	
Purpose:	Returns the calibration factor for the X and potentially the Y axes (bits/mm) as stored in the Lens Config file. This value will also represent the calbration factor for the Y axis if an explicit entry is not present in the Lens Config file.
Implementation:	"GetKFactor" **or** "10"
Parameters	
Returns:	<KFactor> (in floating-point notation)
Comments:	
See also:	

SetPerformanceGlobals (14)

Purpose:	Sets factors to alter the run-time performance of the system.
Implementation:	"SetPerformanceGlobals, <mark-speed-adjust>,<laser-power-adjust>,<pulse-width-adjust>,<pulse-period-adjust>,<orientation>,<X-offset>,<Y-offset>,<Z-offset>" **or** "14,<mark-speed-adjust>,<laser-power-adjust>,<pulse-width-adjust>,<pulse-period-adjust>,<orientation>,<X-offset>,<Y-offset>,<Z-offset>"
Parameters	<mark-speed-adjust> (multiplier for MarkSpeed: 0.5 - 1.5). Specify "NOP" if no change is desired <laser-power-adjust> (multiplier for LaserPower: 0.8 - 1.2). Specify "NOP" if no change is desired <pulse-width-adjust> (multiplier for laser ON pulse width: 0.5 - 1.5). Specify "NOP" if no change is desired <pulse-period-adjust> (multiplier for laser on pulse period: 0.5 - 1.5). Specify "NOP" if no change is desired <orientation> (field orientation in degrees: 0, 90, 180, 270). Specify "NOP" if no change is desired <X-offset> (X axis offset in bits: -32768 - 32767). Specify "NOP" if no change is desired <Y-offset> (Y axis offset in bits: -32768 - 32767). Specify "NOP" if no change is desired <Z-offset> (Z axis offset in bits: -32768 - 32767). Specify "NOP" if no change is desired
Returns:	"0" (command acknowledge)
Comments:	These factors alter the specified marking properties without the need for changing the job. These values are volatile and will not be valid if the EC1000 is reset.
See also:	*ResetPerformanceGlobals*

ResetPerformanceGlobals (15)

Purpose:	Resets the run-time performance modification parameters to their unity values.
Implementation:	"ResetPerformanceGlobals,<persist-to-file>" **or** "15,<persist-to-file>"
Parameters	<persist-to-file> (0 == do not write, 1 == write reset values to the Performance Globals configuration file
Returns:	"0" (command acknowledge)
Comments:	The unity values result in no run-time modification to job specified marking parameters.
See also:	*SetPerformanceGlobals*

OpenCOMPort (16)

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>" **or** "16,<port-ID>,<baud-rate>,<data-bits>,<parity>,<stop-bits>,<flow-control>"
Parameters:	<port-ID> (2 == COM2, 3== COM3) <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:	"0" (command acknowledge)
Comments:	This command is available only if the system is configured for accepting streaming job data over Ethernet. The specified COM port is opened and is available for serial I/O. This operation is intended to permit out-of-band communication 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*

CloseCOMPort (17)

Purpose:	Closes a serial I/O COM port on the EC1000
Implementation:	"CloseCOMPort,<port-ID>" **or** "17,<port-ID>"
Parameters	<port-ID> (2 == COM2, 3 == COM3)
Returns:	"0" (command acknowledge)
Comments:	The COM port is closed and no longer available for serial I/O
See also:	*COMWriteLine, OpenCOMPort*

COMWriteLine (18)	
Purpose:	Writes the string argument to the COM port on the EC1000.
Implementation:	"COMWriteLine,<port-ID>,<string>,<Timeout>" **or** "18,<port-ID>,<string>,<Timeout>"
Parameters:	<port-ID> (2 == COM2, 3 == COM3) <string> (a string to be sent to the COM port) <Time-out> (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 Time-out expires)
Comments:	This operation is intended to permit out-of-band communication 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*

SetMOTFEncoderRate (21)	
Purpose:	Sets the calibration factor used to convert encoder counts to laser galvo command bits (bits/encoderCount)
Implementation:	"SetMOTFEncoderRate,<rate>"
Parameters:	<rate> (-32768.999 - 32768.999 bits/encoderCount)
Returns:	"0" (command acknowledge)
Comments:	The encoder rate relates encoder counts to how far an object travels in the lens field in galvo command bits.
See also:	

GetZKFactor (27)	
Purpose:	Returns the calibration factor for the Z axis (bits/mm) as stored in the Lens Config file
Implementation:	"GetZKFactor" **or** "27"
Parameters	
Returns:	<ZKFactor> (in floating-point notation)
Comments:	
See also:	

GetYKFactor (28)	
Purpose:	Returns the calibration factor for the Y axes (bits/mm) as stored in the Lens Config file. If the entry is missing from the Lens Config file, the value will be the same as returned by the *GetKFactor* command.
Implementation:	"GetYKFactor" **or** "28"
Parameters	
Returns:	<KFactor> (in floating-point notation)
Comments:	
See also:	

GetControllerTemp (29)	
Purpose:	Returns the current temperature of the EC1000 board in degrees Celsius
Implementation:	"GetControllerTemp" **or** "29"
Parameters	
Returns:	<temp> (in floating-point notation, degrees C)
Comments:	
See also:	

6.3.2 Job Execution Control

GetFlashJobFileList (203)

Purpose:	Returns a comma separated list of job files located on the Flash file system located on the EC1000
Implementation:	"GetFlashJobFileList" **or** "203"
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*

GetUSBJobFileList (204)

Purpose:	Returns a comma separated list of job files located on the USB Flash file system attached to the EC1000
Implementation:	"GetUSBJobFileList" **or** "204"
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*

LoadFlashJob (205)

Purpose:	Loads a job from the EC1000 resident Flash file system
Implementation:	"LoadFlashJob,<job-name>" **or** "205,<job-name>"
Parameters:	<job-name> (the name of a job stored on the EC1000)
Returns:	"0" (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*

LoadUSBJob (206)

Purpose:	Loads a job from the USB Flash file system attached to the EC1000
Implementation:	"LoadUSBJob,<job-name>" **or** "206,<job-name>"
Parameters:	<job-name> (the name of a job stored on the USB Flash file system device)
Returns:	"0" (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*

ExecuteJobOnce (207)	
Purpose:	Starts the execution of a job one time
Implementation:	"ExecuteJobOnce, <job-name>" **or** "207, <job-name>"
Parameters	<job-name> (must be one of the jobs loaded with LoadFlashJob or LoadUSBJob)
Returns:	"0" (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 it was constructed with a <WaitFoIO> instruction. The job can be stopped 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*

ExecuteJobContinuous (208)	
Purpose:	Starts the execution of a job and repeats it forever
Implementation:	"ExecuteJobContinuous, <job-name>" **or** "208, <job-name>"
Parameters:	<job-name> (must be one of the jobs loaded with LoadFlashJob or LoadUSBJob)
Returns:	"0" (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 is required 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*

GetJobStatus (209)	
Purpose:	Returns the status of the currently executing job
Implementation:	"GetJobStatus" **or** "209"
Parameters:	
Returns:	"Idle" (no job is executing; a job may or may not be loaded) "Busy" (a job is executing)
Comments:	
See also:	

GetJobState (211)	
Purpose:	Returns the state of the currently executing job
Implementation:	"GetJobState" **or** "211"
Parameters:	
Returns:	<current-sequence-index> The index number of the currently executing sequence <current-sequence-count> Numer of iterations of the current executing sequence <current-segment-index> The index number of the currently executing segment <current-segment-count> Number of iterations of the currentl executingsegment <current-segment-name> The name of the currently executing segment Ex: "1,2,3,1,Preamble"
Comments:	
See also:	

GetJobElapsedTime (212)	
Purpose:	Returns the last measured duration in milli-seconds of the currently executing job
Implementation:	"GetJobElapsedTime" **or** "212"
Parameters:	
Returns:	<time-in-msec> Last measured job execution duration in milli-seconds
Comments:	Time is measured based in tke monitoring of the BeginJob and EndJob events. Jobs must be constructed with these instructions to be measured.
See also:	

6.3.3 System Administration Commands

SetAdminPIN (500)	
Purpose:	Sets the Administrator PIN (password)
Implementation:	"SetAdminPIN,<admin-pin>" **or** "500,<admin-pin>"
Parameters:	<admin-pin> (new administrator PIN as a numeric string)
Returns:	"0" (command acknowledge)
Comments:	The host must have exclusive control of the EC1000 (*TakeHostControl*) before issuing this command. The Administrator PIN is used with the Pendant interface to protect access to administrator functions
See also:	*SetAdminPIN, GetUserPIN, SetUserPIN*

GetAdminPIN (501)	
Purpose:	Gets the current Administrator PIN (password)
Implementation:	"GetAdminPIN" **or** "501"
Parameters:	
Returns:	<admin-pin> (Administrator PIN as a numeric string)
Comments:	The Administrator PIN is used with the Pendant interface to protect access to administrator functions
See also:	*SetAdminPIN, GetUserPIN, SetUserPIN*

SetDHCPMode (502)	
Purpose:	Sets the DHCP addressing mode
Implementation:	"SetDHCPMode,<mode>" **or** "502,<mode>"
Parameters	<mode> ("Static" or "Autodetect")
Returns:	"0" (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 time-out 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*

GetDHCPMode (503)	
Purpose:	Gets the current DHCP addressing mode
Implementation:	"GetDHCPMode" **or** "503"
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 time-out 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*

SetLocalGateway (504)	
Purpose:	Sets the gateway IP address used by the EC1000 if in static IP addressing mode
Implementation:	"SetLocalGateway,<gateway-address>" **or** "504,<gateway-address>"
Parameters:	<gateway-address> (in dot notation, e.g. 192.168.101.2)
Returns:	"0" (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*

GetLocalGateway (505)	
Purpose:	Returns the gateway IP address used by the EC1000 if in static IP addressing mode
Implementation:	"GetLocalGateway" **or** "505"
Parameters:	
Returns:	<gateway-address> (in dot notation, e.g. 192.168.101.2)
Comments:	
See also:	*SetLocalGateway*

SetLocalIP (506)	
Purpose:	Sets the IP address used by the EC1000 if in static IP addressing mode
Implementation:	"SetLocalIP,<IP-address>" **or** "506,<IP-address>"
Parameters:	<IP-address> (in dot notation, e.g. 192.168.101.200)
Returns:	"0" (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*

GetLocalIP (507)	
Purpose:	Returns the IP address used by the EC1000 if in static IP addressing mode
Implementation:	"GetLocalIP" **or** "507"
Parameters:	
Returns:	<static-IP-address> (in dot notation, e.g. 192.168.101.2)
Comments:	
See also:	*SetLocalIP*

SetNodeFriendlyName (508)	
Purpose:	Sets the "friendly name" of the EC1000
Implementation:	"SetNodeFriendlyName,<friendly-name>" **or** "508,<friendly-name>"
Parameters:	<friendly-name> (string representing the friendly name assigned to the EC1000)
Returns:	"0" (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*

GetNodeFriendlyName (509)	
Purpose:	Returns the "friendly name" of the EC1000
Implementation:	"GetNodeFriendlyName" **or** "509"
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*

SetSubnetMask (510)	
Purpose:	Sets the subnet mask used by the EC1000 if in static IP addressing mode
Implementation:	"SetSubnetMask,<mask>" **or** "510,<mask>"
Parameters:	<mask> (in dot notation, e.g. 255.255.255.0)
Returns:	"0" (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*

GetSubnetMask (511)	
Purpose:	Returns the subnet mask used by the EC1000 if in static IP addressing mode
Implementation:	"GetSubnetMask" **or** "511"
Parameters	
Returns:	<subnet-mask> (in dot notation, e.g. 255.255.255.0)
Comments:	
See also:	*SetSubnetMask*

SetUserPIN (512)	
Purpose:	Sets the Administrator PIN (password)
Implementation:	"SetUserPIN,<user-pin>" **or** "512,<user-pin>"
Parameters	<user-pin> (new user PIN as a numeric string)
Returns:	"0" (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*

GetUserPIN (513)	
Purpose:	Gets the current User PIN (password)
Implementation:	"GetUserPIN" **or** "513"
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*

SetCOMPortSpeed (514)	
Purpose:	Sets the speed of the pendant, api and motion-control COM ports
Implementation:	"SetCOMPortSpeed,<pendant-port-baud-rate>,<api-port-baud-rate>,<motion-control-port-baud-rate>" **or** "514,<pendant-port-baud-rate>,<api-port-baud-rate>,<motion-control-port-baud-rate>"
Parameters:	<pendant-port-baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000}) <api-port-baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000}) <motion-control-port-baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000})
Returns:	"0" (command acknowledge)
Comments:	The three COM ports on the EC100 are logically identified as “pendant”, “api” and “motion-control” and are physically mapped using the command *SetCOMPortAssignments.*
See also:	*GetCOMPortSpeed, SetCOMPortAssignments, GetCOMPortAssignments*

GetCOMPortSpeed (515)	
Purpose:	Gets the current speed of the pendant, api and motion-control COM ports
Implementation:	"GetCOMPortSpeed" **or** "514"
Parameters:	<pendant-port-baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000}) <api-port-baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000}) <motion-control-port-baud-rate> (one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000})
Returns:	<pendant-port-baud-rate>,<api-port-baud-rate>,<motion-control-port-baud-rate> (each value can be one of {110,300,1200,2400,4800,9600,19200,38400,57600,115200,128000,256000})
Comments:	
See also:	*SetCOMPortSpeed, SetCOMPortAssignments, GetCOMPortAssignments*

SetCOMPortAssignments (516)	
Purpose:	Maps the EC1000 COM ports to the logical pendant, api and motion-control ports
Implementation:	"SetCOMPortAssignments,<pendant-port>,<api-port>,<motion-control-port>" **or** "516,<pendant-port>,<api-port>,<motion-control-port>"
Parameters:	<pendant-port> (1, 2, or 3) <api-port> (1, 2, or 3) <motion-control-port> (1, 2, or 3)
Returns:	"0" (command acknowledge)
Comments:	The COM port assignments must be unique. If hardware flow control is required, then COM1 (1) should be used. This command updates the contents of the AdminConfig file.
See also:	*SetCOMPortSpeed, GetCOMPortSpeed, GetCOMPortAssignments*

GetCOMPortAssignments (517)	
Purpose:	Gets the current mapping of the EC1000 COM ports to the logical pendant, api and motion-control ports
Implementation:	"GetCOMPortAssignments>" **or** "517"
Parameters:	
Returns:	"<pendant-port>,<api-port>,<motion-control-port>" (1, 2, or 3)
Comments:	
See also:	*SetCOMPortSpeed, GetCOMPortSpeed, SetCOMPortAssignments*

6.4 Remote control return codes

In certain cases, the response messages may be an error message rather than the expected “0” (ACK) or return variable(s). The following table defines the possible codes that may be returned.

Code	Meaning	Description
0	Success	Command processed with no error
8	Bad Command	The command was not recognized
9	Bad Arg	The command passed inappropriately formed arguments, or no argument if an argument was required
100	No Files Found	The named job file was not found
101	No Drive	No USB disk drive was found
106	Not In Host Control	The command required that exclusive control of the EC1000 be obtained (*TakeHostControl*)
107	Wrong Host Type	Serial port I/O is only allowed while the streaming LAN interface is in control
108	Error Job Busy	Command cannot execute a job is running
110	Error Software	An internal software exception occurred
206	Cannot Create Port	An error occurred while trying to open a COM port for serial communications
207	Cannot Open Port	Cannot open the serial port
208	Port Not Open	Serial port was not opened before the requested command
209	Port Timeout	The serial port timed-out waiting for input
210	Wrong Port Number	An invalid COM port ID was specified

Appendix 1 Theory Of Operation

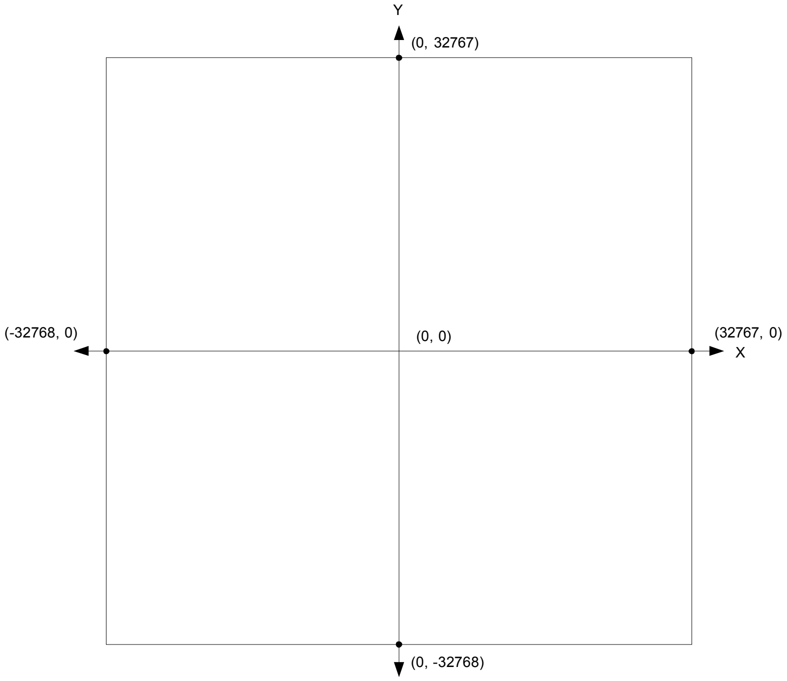
A1.1 Scanning Job Fundamentals

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".

A1.1.1 Coordinate system conventions

Both of the basic movement commands, "mark" and "jump" are expressed in a cartesian coordinate system that is illustrated in Figure 11:

Figure 11: 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.

A1.1.2 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 12: Laser marking sample.

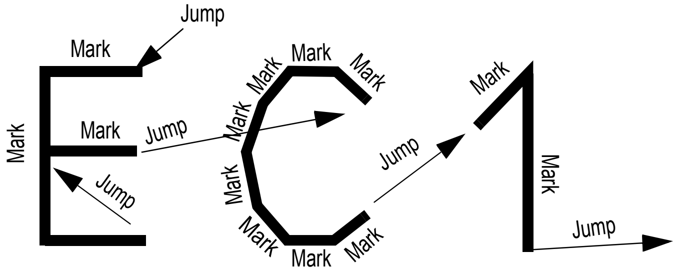


Figure 12: shows a sample of the beginning of a simple laser marking. The image is composed of straight line segments (vectors). Connected line segments are formed with sequential *Mark* commands and spaces between unconnected segments are formed with *Jump* commands. Both *Marks* and *Jumps* are controlled-velocity 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 12:, 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.

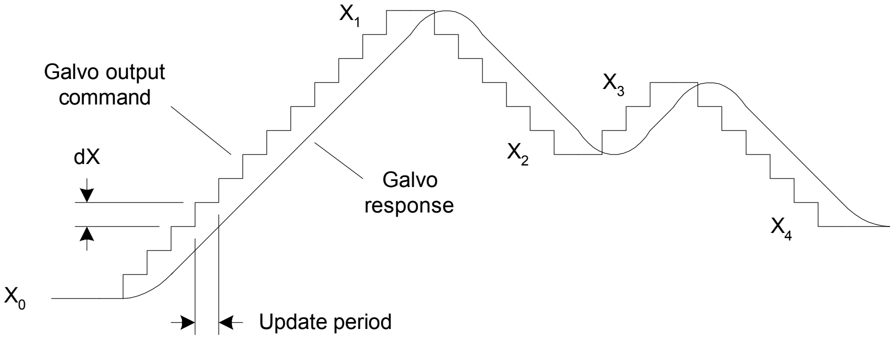
The initial galvanometer position after system power-up is the center of the image field. Marks and jumps are specified from the current position of the galvanometers to a new target position. Jobs typically begin with an absolute jump to the first marking position, and after that, each vector (jump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

A1.1.3 Micro-vectoring

Controlled velocity marking and jumping is accomplished through a process call micro-vectoring. This process is illustrated in the Figure 13: 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.

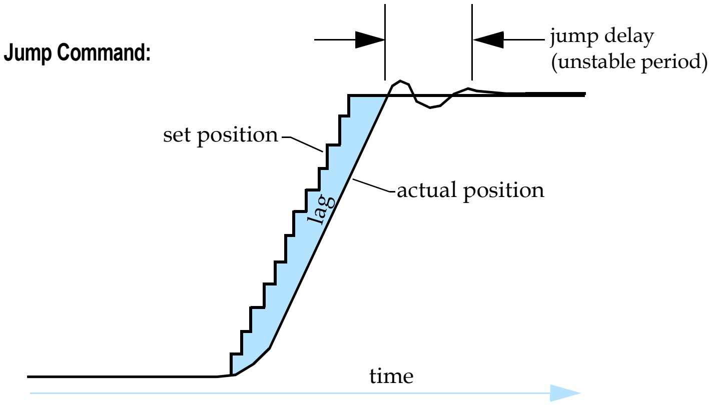
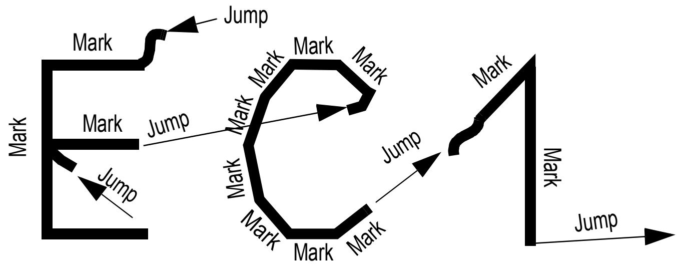
Figure 13: Micro-vector operation

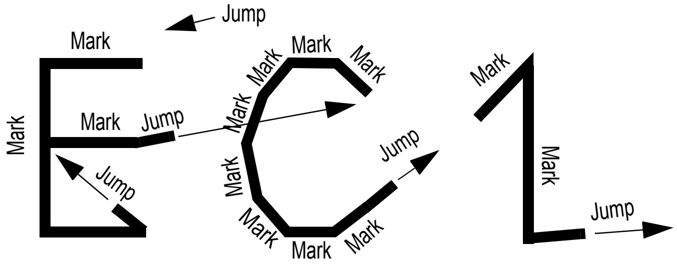
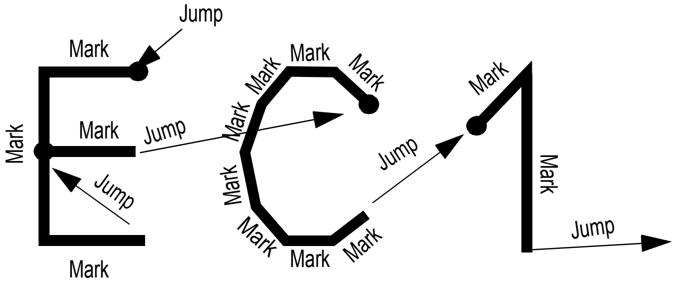
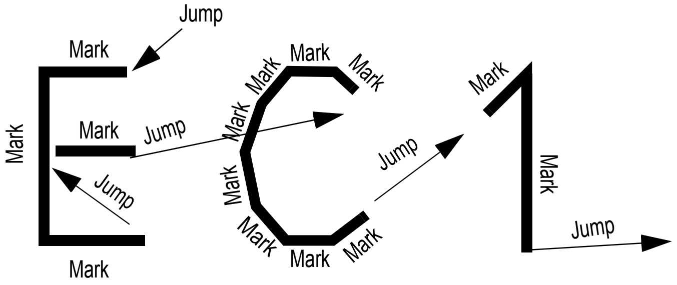
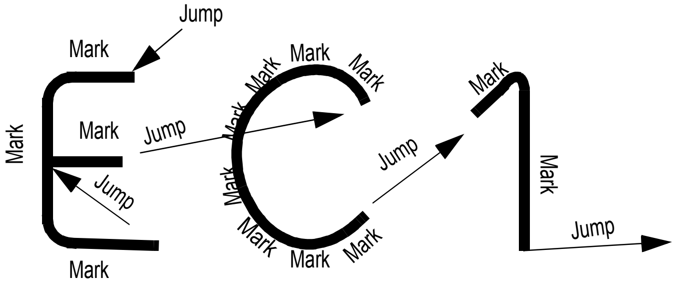
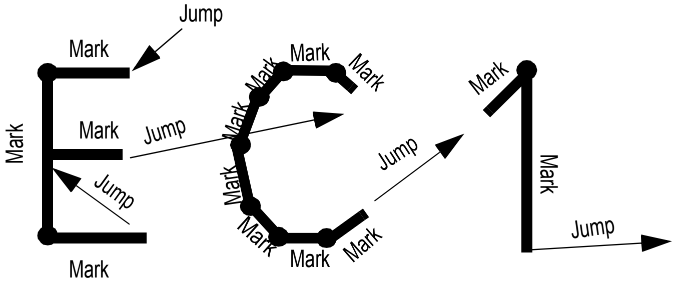


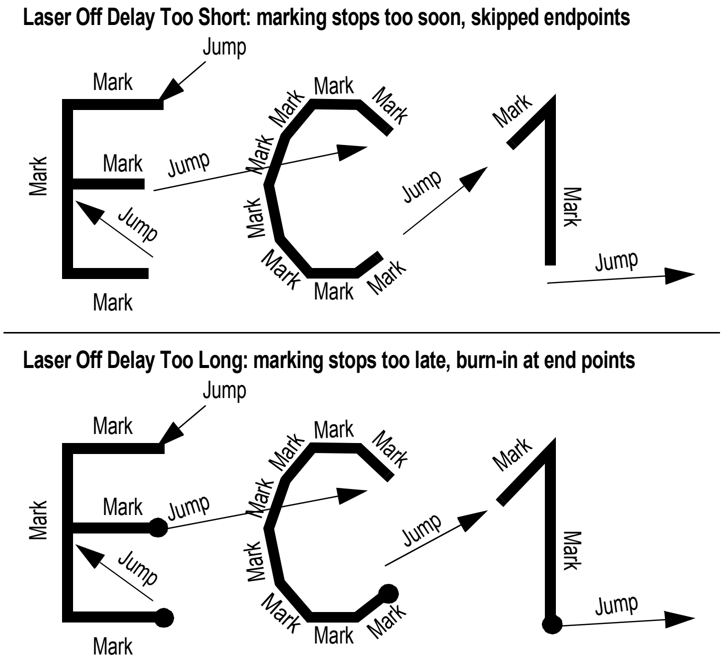
A1.1.4 Delays

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.

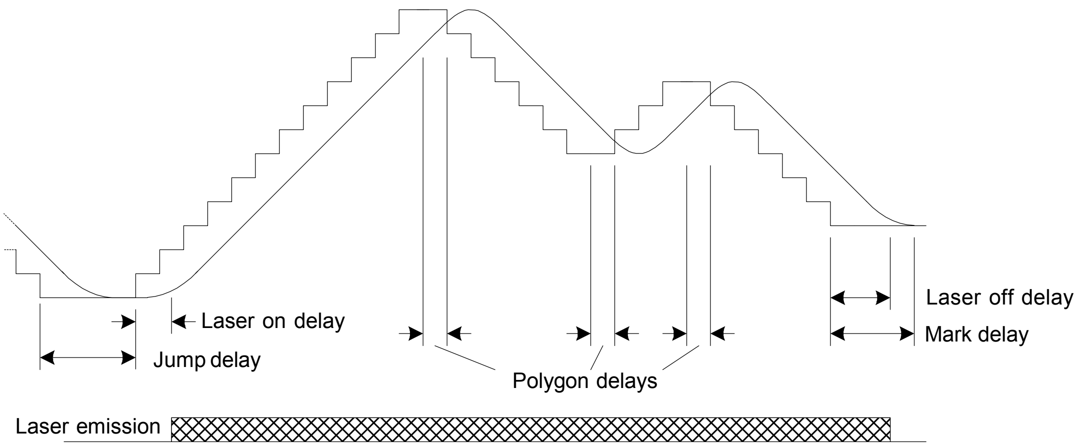
Parameter	Purpose	Effects
JumpDelay	During a jump, the system mirrors accelerate to rapidly get to the next mark position, ideally at the fastest speed possible to minimize overall marking time. As with all accelerations, mirror and system inertia create a slight lag at the beginning of the acceleration. 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 lag actual position time Jump Delay Too Short: Marking starts before mirrors properly settle  Jump Mark Mark Mark Mark Mark Mark Jump Mark Mark Jump Jump 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 Jump Mark Mark 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 Mark Mark Mark Jump Mark Jump Mark Jump Mark Mark Mark Mark Jump Mark **Laser On Delay Too Long: marking starts too late, skips points**  Jump Mark Mark Mark Mark Mark Mark Jump Mark Jump Mark Jump Mark Mark Mark Mark Jump Mark
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 Mark Mark Mark Jump Mark Jump Mark Jump Mark Mark Mark Mark Jump Mark **Polygon Delay Too Long: burn-in at junctions in the vector**  Jump Mark Mark Mark Mark Mark Mark Jump Mark Jump Mark Jump Mark Mark Mark Mark Jump Mark

LaserOffDelay	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. The goal is to adjust the Laser Off Delay to ensure uniform marking with no variations of intensity throughout the desired vector. 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.	

The relationship of the delays to the micro-vectoring process is illustrated in Figure 14:.

Figure 14: Micro-vectoring and laser timing relationships

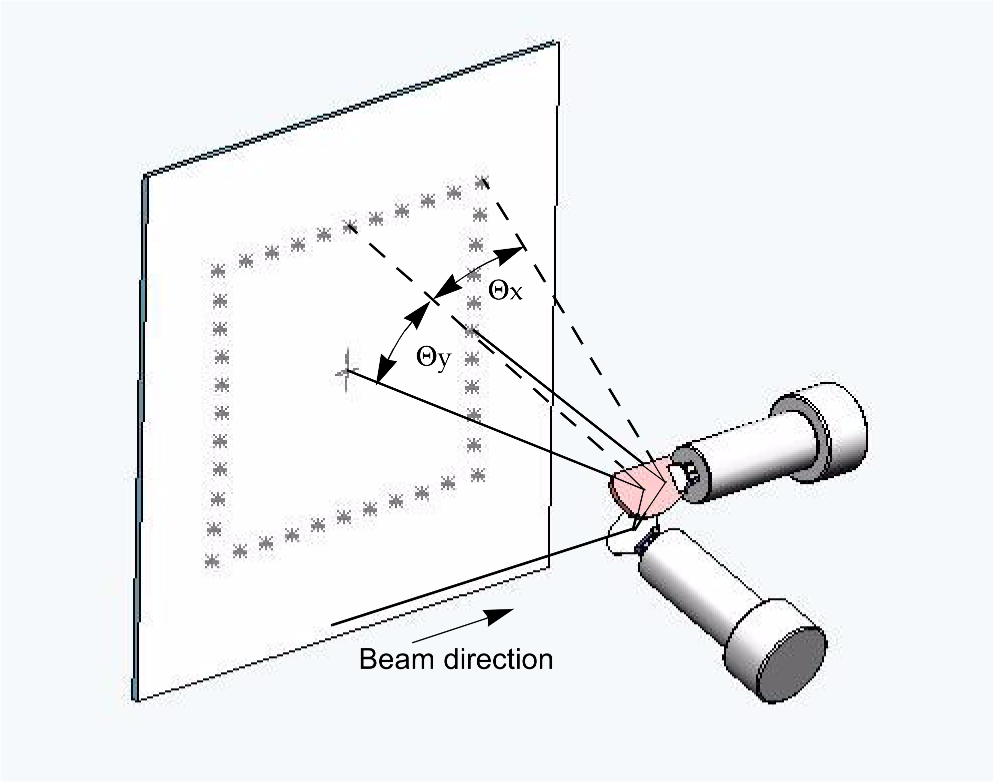


A1.2 Image Field Correction

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 15: shows the basic projection system layout.

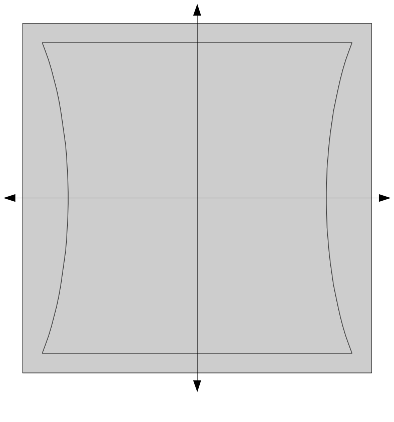
Figure 15: Projection system layout



A1.2.1 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.

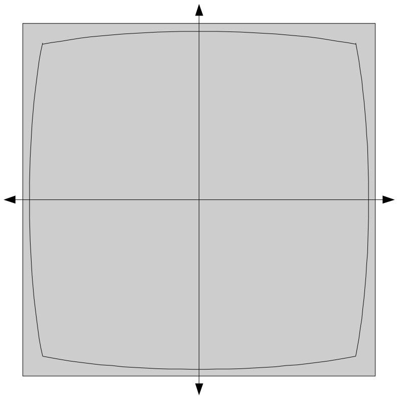
Figure 16: Pincushion distortion caused by X-Y mirror set



A1.2.2 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 17:, 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

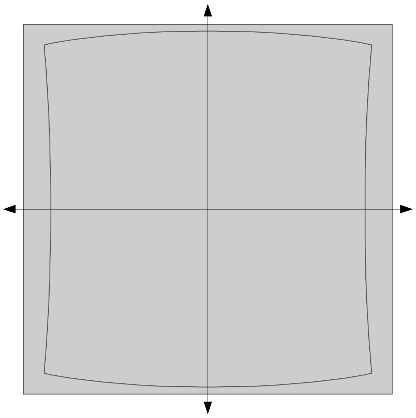
Figure 17: Pillow distortion caused by F-theta lens.



A1.2.3 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 18:.   This distortion is automatically compensated for by the EC1000 through the use of correction tables.

Figure 18: 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".

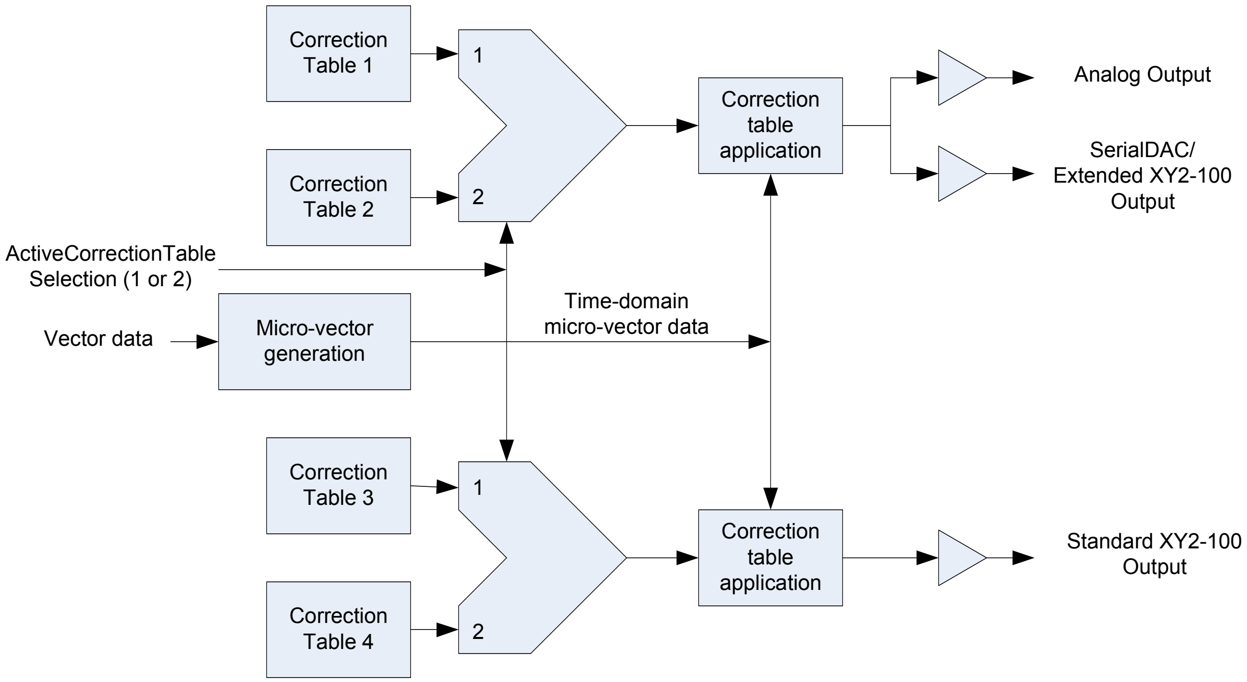
During the micro-vectoring process at each update interval, the EC1000 calculates the ideal position of the mirrors along the path. It compares this value to the correction table grid and accesses the four grid points that immediately surround the calculated point. The corrections at these four points 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.

A1.2.4 Multiple correction table support

The EC1000 has integral support for up to four independent three-axis correction tables. These tables are organized in pairs where the first pair is applied to the analog D/A converters, and the second is applied to the XY2-100 port. Which table of each pair that is actually used is dynamically selectable though the job parameter *ActiveCorrectionTable*. The first of the two tables in the pair is intended to be used when actual laser processing is taking place. The second table of the pair is intended to be used with a pointer laser.

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.

Figure 19: Multiple correction table usage in the EC1000

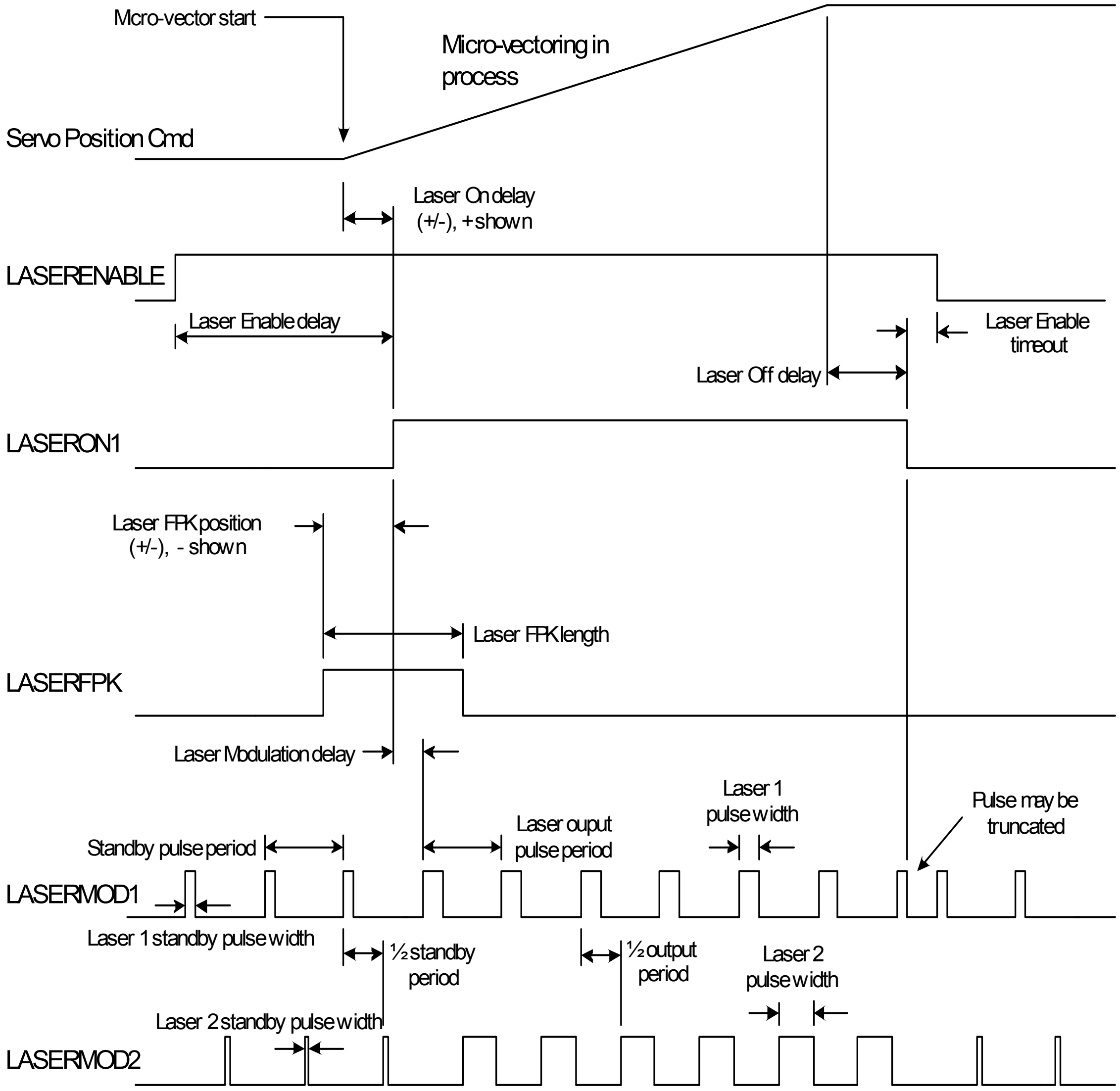

Correction Table 1
Correction Table 2
1
2
Correction table application
Analog Output
SerialDAC/ Extended XY2-100 Output
ActiveCorrectionTable Selection (1 or 2)
Vector data
Micro-vector generation
Time-domain micro-vector data
Correction Table 3
Correction Table 4
1
2
Correction table application
Standard XY2-100 Output

A1.3 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 1usec.

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

Figure 20: Laser timing relationships.



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 20: 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.

A1.4 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 *Table 17: Example IPG Fiber Laser Configuration XML.*.

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 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 10: 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.
<LsrEnaDly>**7**</LsrEnaDly>	<set id=”LaserEnableDelay”>**7**</set>	Wait 7msec after asserting the LASERENABLE signal
<LsrEnaTmo>**4**</LsrEnaTmo>	<set id=”LaserEnableTimeout”>**4**</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

Static Configuration XML	**Dynamic Configuration XML**	**Example Description**
<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

A1.4.1 Laser Timing Emulation

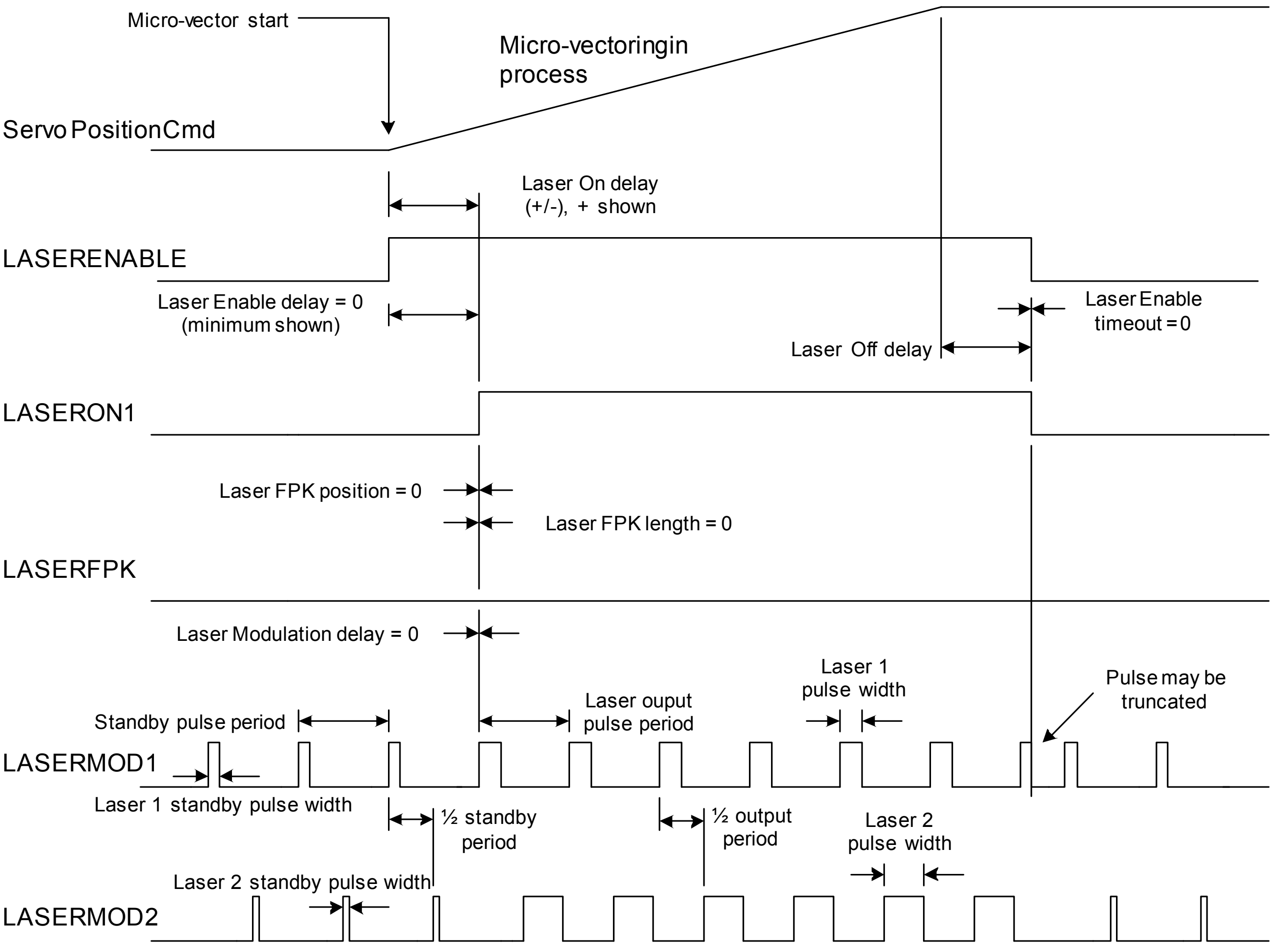
Traditional laser scanning controllers often use fixed signal sets and constrained timing relationships to provide laser control, whereas the EC1000 uses a completely flexible and programmable suite of signals. The EC1000 can be configured to emulate the timing produced be other commercial controllers 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 21: Laser timing for CO_2 laser systems



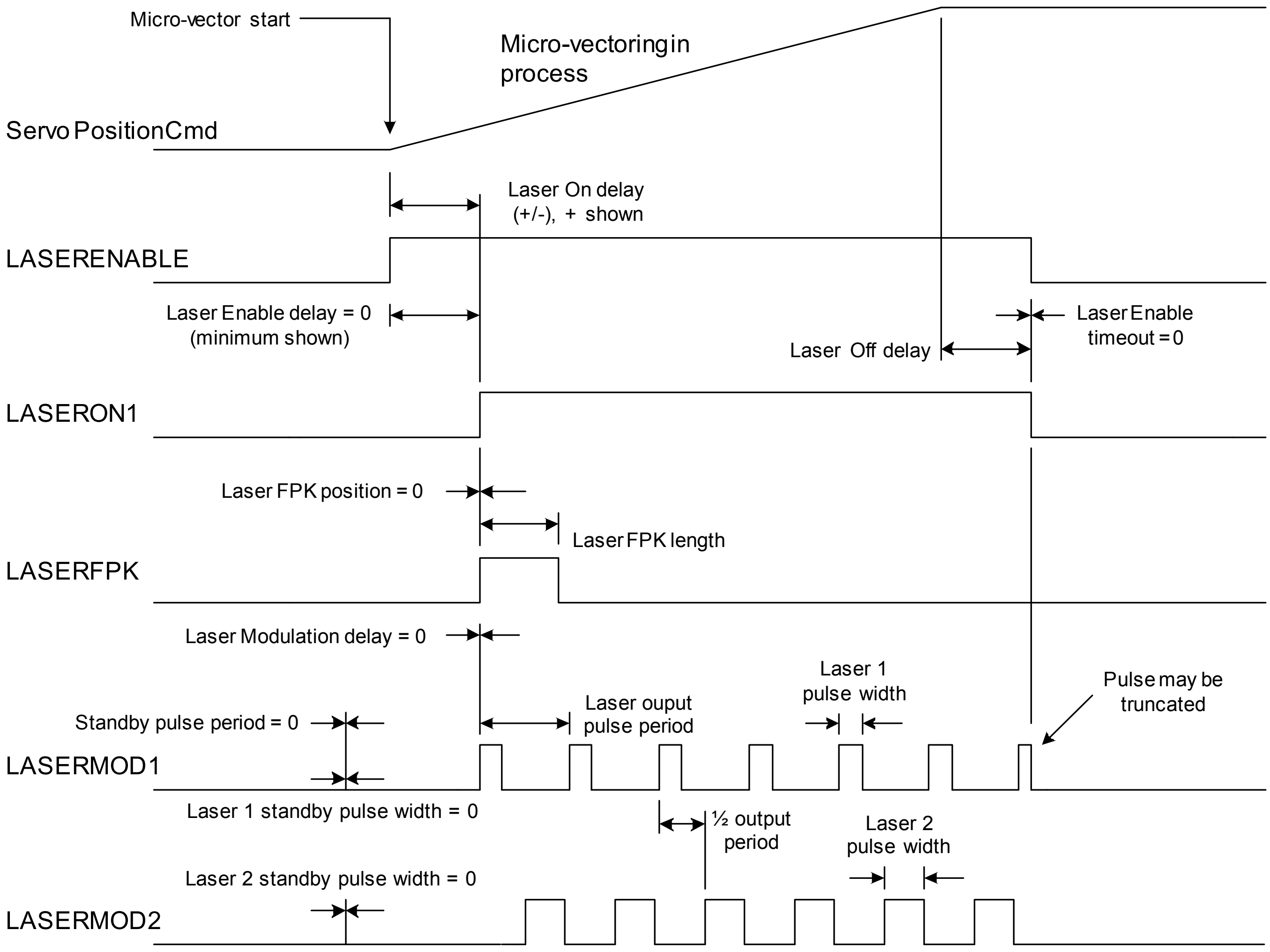
The simplest emulation mode is for CO_2 lasers. These lasers do not require a Laser FPK signal so these parameters are set to zero. LASERENABLE is also not typically needed therefore the Laser Enable delay and Laser Enable timeout can be set to zero to maximize throughput. In fact, whenever LASERENABLE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

Table 11: 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 22: Nd:YAG Emulation Mode-1 (Raylase Nd:YAG Mode-1 and Scanlab YAG 1)



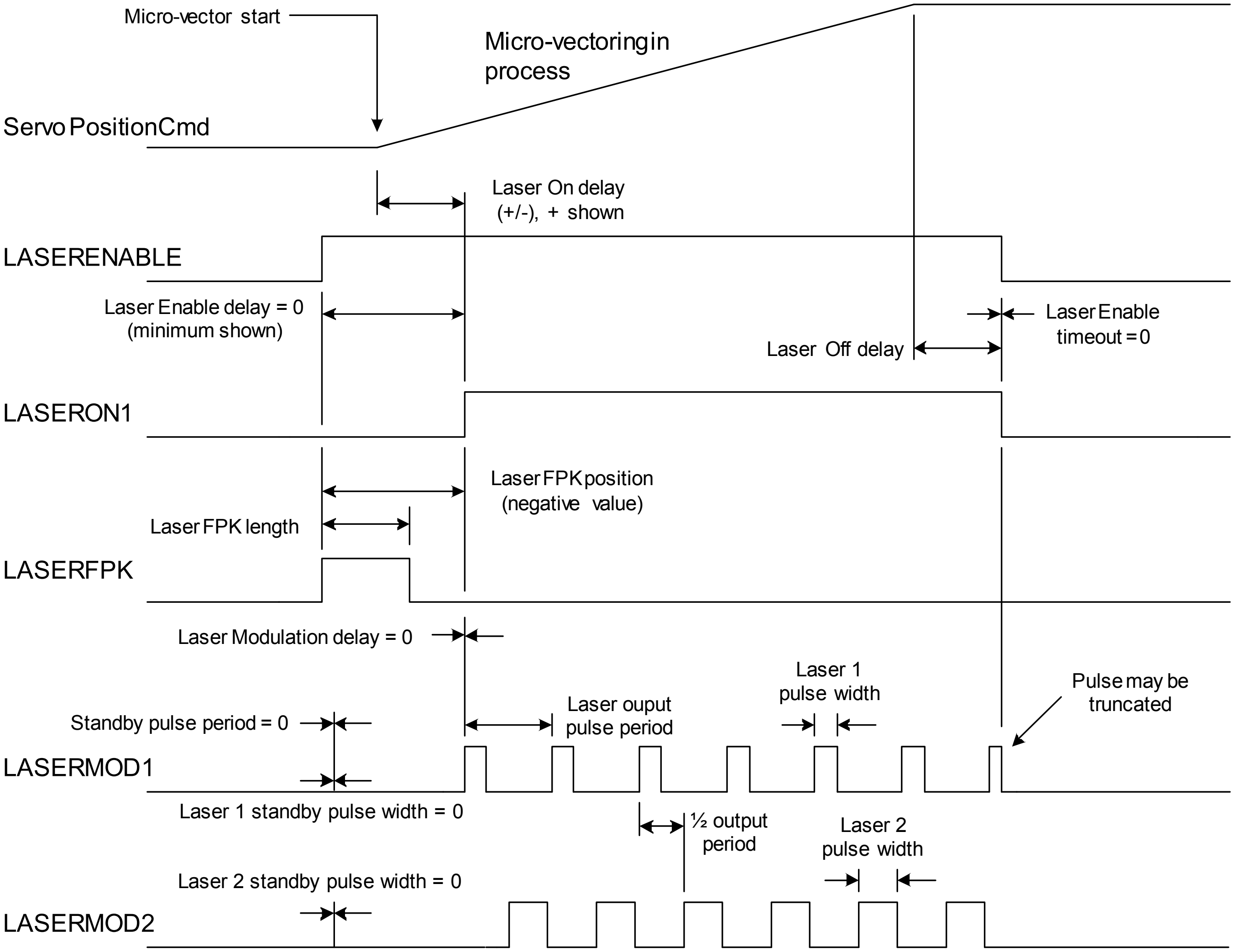
Most of theYAG modes do not require standby or idle pulses. To supress these pulses, the Standby pulse width and pulse period are set to zero. In this mode, the LASERFPK is asserted coincident with the LASERON and LASERMOD signals, but its assertion can have variable length. If the Laser On delay is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

Table 12: 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 23: Nd:YAG Emulation Mode-2 (Raylase Nd:YAG Mode-2)



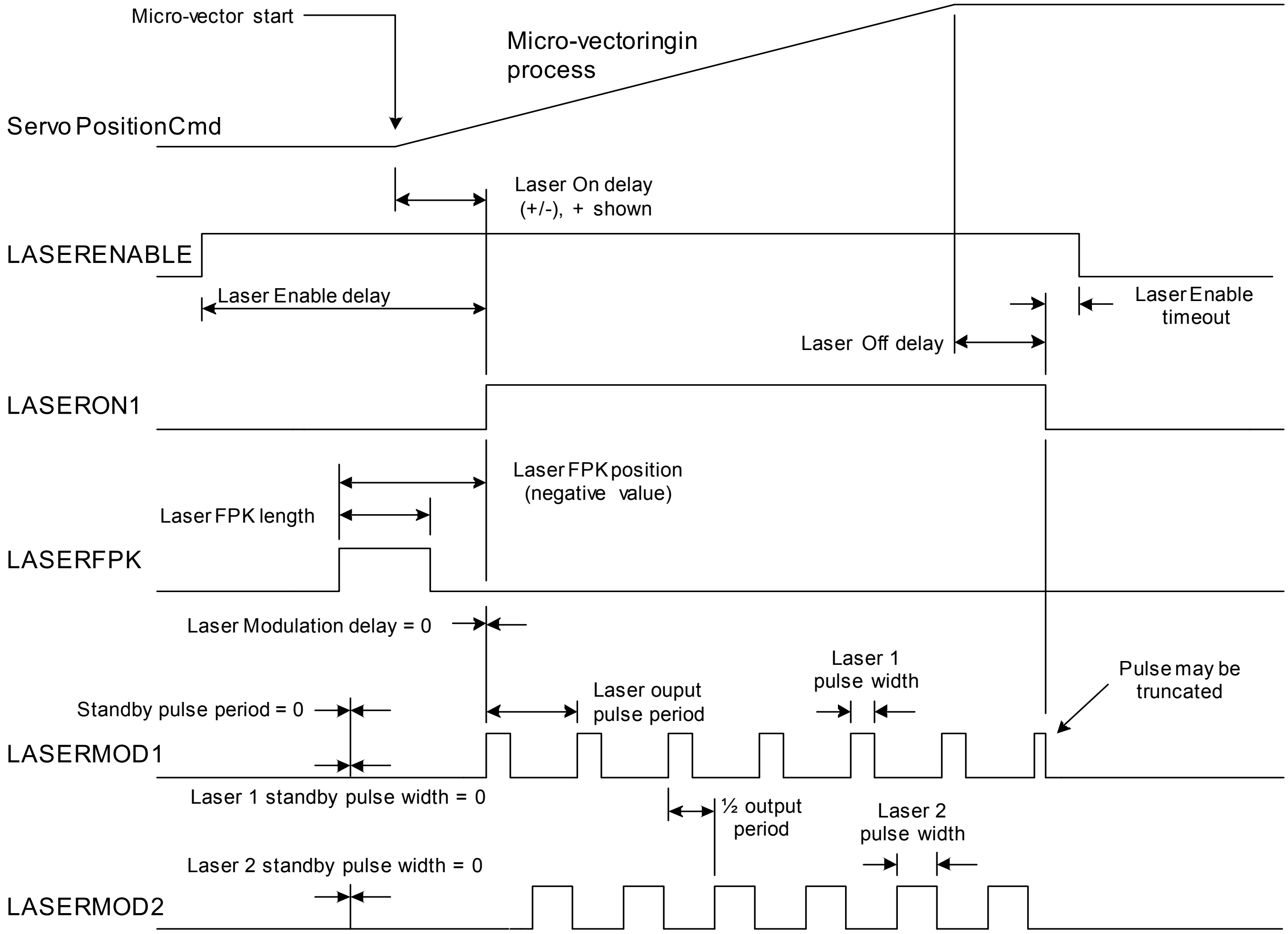
In this mode, the LASERFPK signal is a 10µ sec pulse asserted a variable amount of time prior to the assertion of LASERON and the coincident generation of pulses. This timing is typically suited for Lee and Coherent lasers.

Table 13: 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 24: Nd:YAG Emulation Mode-3 (Raylase Nd:YAG Mode-3)



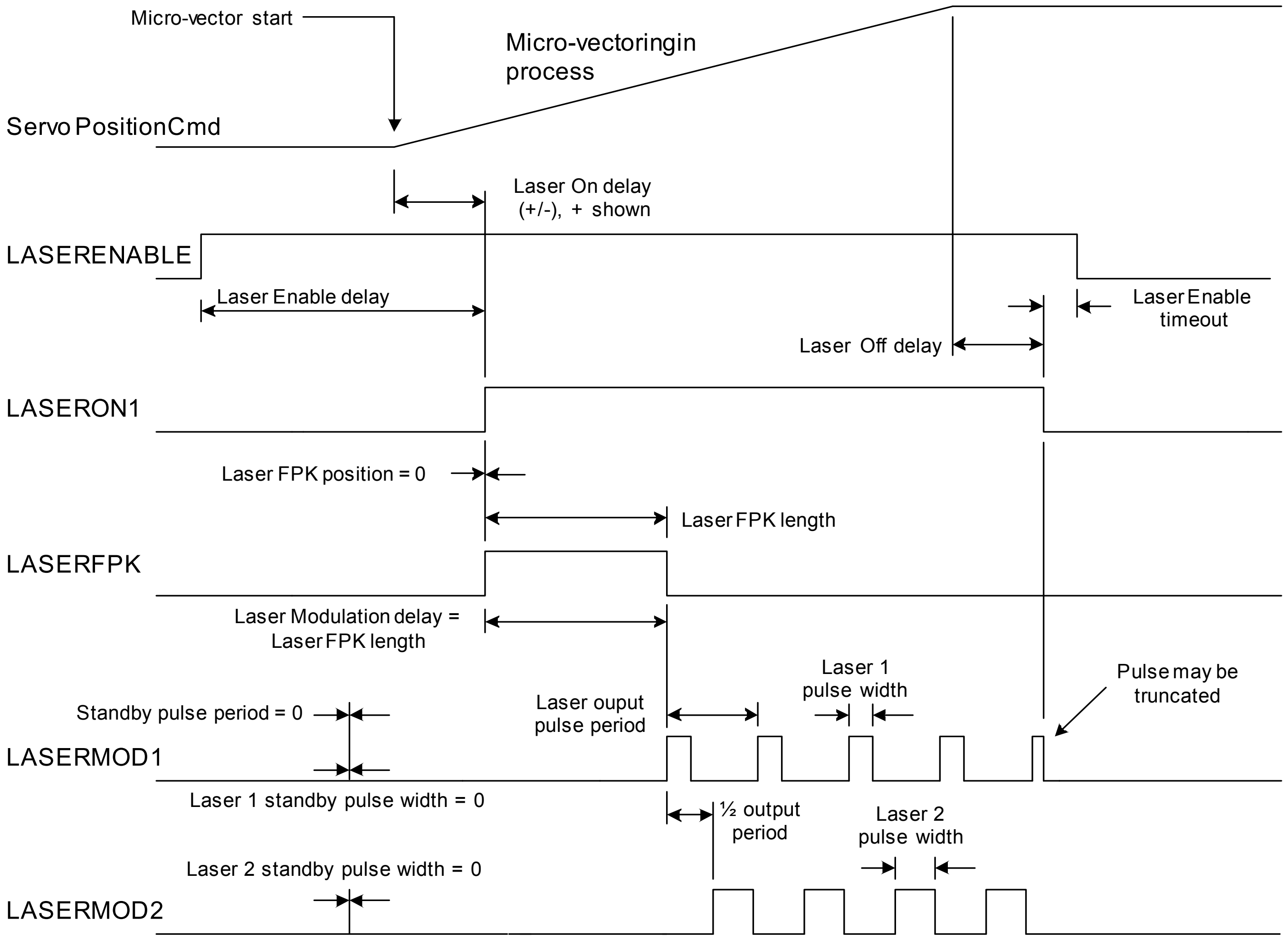
This mode is very similar to Mode-2. The difference is that Laser FPK length can vary. Spectron lasers normally use this type of timing.

Table 14: 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 25: Nd:YAG Emulation Mode-4 (Scanlab YAG 2)



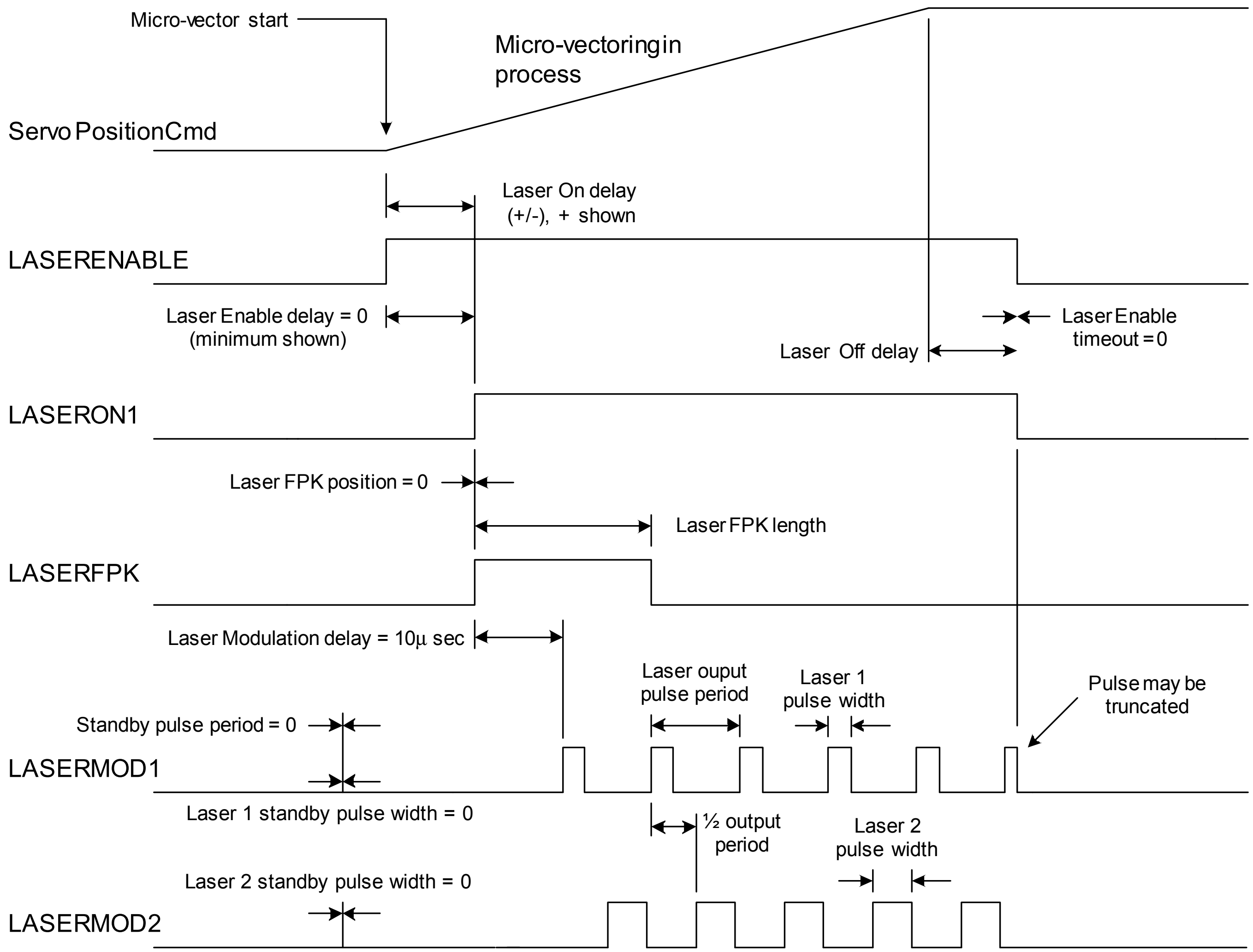
In this mode, the LASERFK signal leading edge is coincident with the leading edge of LASERON and the generation of the laser pulses is delayed to be coincident with the trailing edge of the LASERFPK signal.

Table 15: 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 26: Nd:YAG Emulation Mode-5 (Scanlab YAG 3)



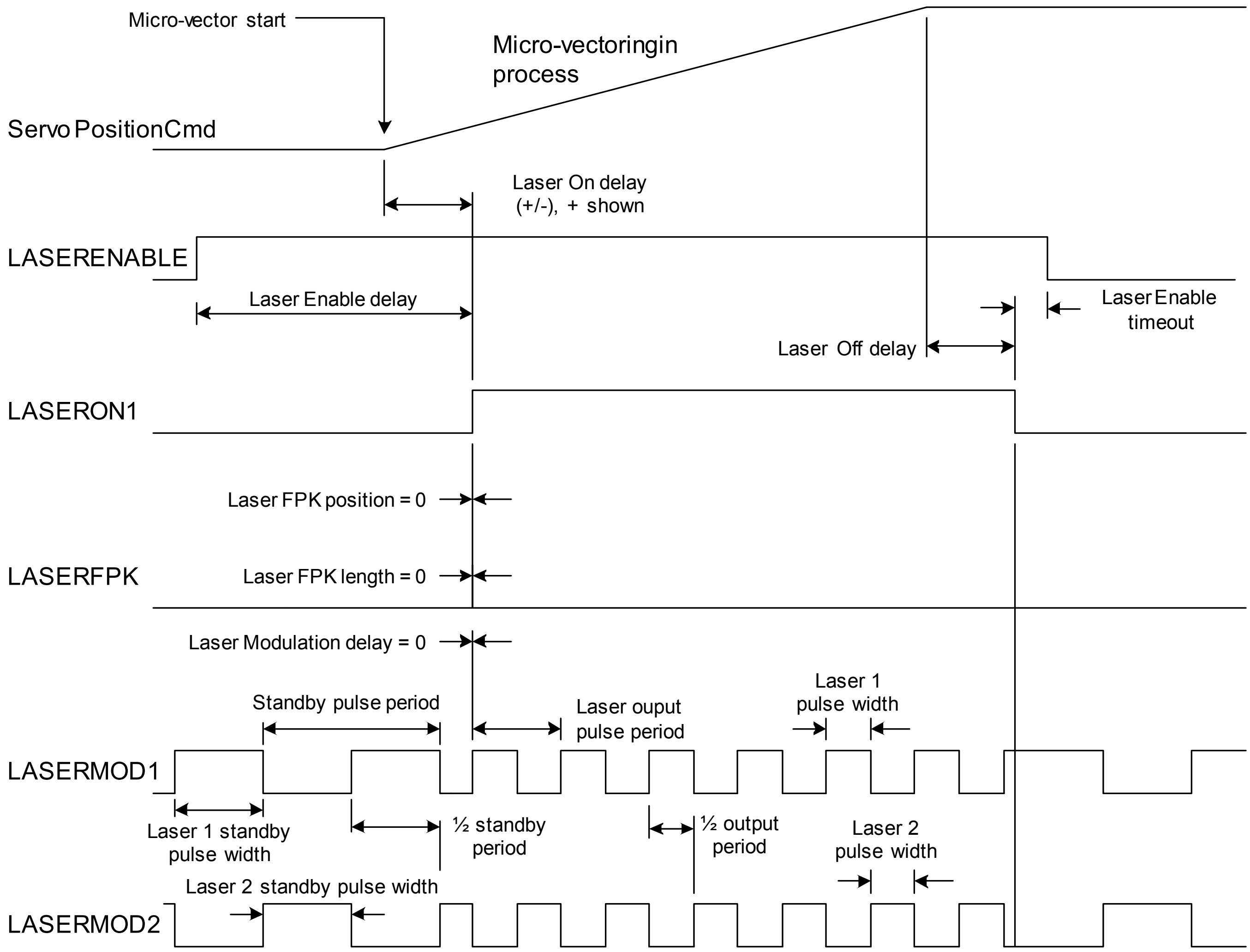
This mode is very similar to emulation mode-4. The difference is that the start of laser pulse generation is 10µ sec after the coincident leading edges of LASERON and LASERFPK.

Table 16: 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>	<set id="LaserFPK">**0, 20**</set>	Example FPK length set to 20µsec with no shift relative to LASERON
<FpsWidth>**20**</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

Fiber Laser Timing

Figure 27: 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 Type A interface 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 17: Example IPG Fiber Laser Configuration XML.

Static Configuration XML	Dynamic Configuration XML	Example Description
<LsrEnaDly>**7**</LsrEnaDly>	<set id=”LaserEnableDelay”>**7**</set>	Minimum master oscillator startup time
<LsrEnaTmo>**10**</LsrEnaTmo>	<set id=”LaserEnableTimeout”>**10**</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.