The Green Bank Telescope
Laser Ranging System
ZY Software
Reference Manual

Ramon Creager

January 24, 1994

Contents

Chapter 1

Introduction

The Green Bank Telescope Laser Ranging System will be made up of approximately 20 laser ranging instruments to be used for active surface and precision pointing. These instruments are identified as ZY001—ZY020. Groups of ZYs will be under the control of computers identified as ZIYS for active surface control, and ZIYP for precision pointing.

Under normal operation, the active surface, precision pointing, and monitor and control interface to the Laser Ranging System will be through the ZIYs. The ZIYs will be responsible for coordinating the operation of the ZYs to obtain 3-D coordinates of retroreflectors on the GBT.

This document is a draft of the software reference manual for the ZY instruments—and thus primarily of interest to the ZIY software designer. It does, however, give others a basic understanding of the ZY system operation, and therefore a fundamental understanding of potential capabilities of the ZIYs.

It should be noted that the list of functions includes only the completed and tested functions. A number of additional functions are under development—particularly in areas of tracking and time synchronisation.

Comments and questions are solicited on both the content and format of this draft, so corrections and enhancements can be made to the final document. Companion software reference documents on the ZIYS and ZIYP systems will be released in the future. ZY hardware and calibration reference manuals will also be released in the future.

Chapter 2

Installing the ZY systems

2.1 Requirements

To use the ZY systems, the following hardware and software is required:

- Hardware:
  1. ISA bus computer with at least an Intel 286 microprocessor, with at least 640K of memory.
  2. Intel 287 or better math co-processor.
  3. At least 1 floppy disk (other storage media optional).
  4. Any display adapter (the computer will not run without one).
  5. Quatech DAQ-16 100 KHz A/D converter, at I/O address 240 Hex, IRQ disabled, DMA channel 5. This board must be modified as per drawing D35420S002.
  6. MICRO RC 5328 Motion Control Card, at I/O 100 Hex, IRQ disabled.
  7. 3COM Etherlink II or Etherlink III ethernet adapter, at I/O 330 Hex, IRQ 5.
  8. A connection to a TCP/IP LAN.
- Software:
  1. MS-DOS 5.0 or greater.
  2. FTP Software PC/TCP for DOS version 2.1 or greater.

2.2 Installation Steps

The software installation of the ZY system consists of the following steps:

- Creating a boot diskette that contains the nescessary files.
- Establishing a network account that the ZY system can use to retrieve the needed program and initialization files.

2.2.1 Boot Diskette

SYSTEM DISKETTE

Create a system diskette (example: format a: /s), with MS-DOS 5.0.

CONFIG.SYS

Create a CONFIG.SYS file as follows:

```
device=a:\pctcp\protman.sys /i:a:\pctcp
device=a:\pctcp\elnkii.dos
device=a:\pctcp\dis_pkt.gup
files=10
buffers=20
```

The lines in the CONFIG.SYS file have the following meaning:

`device=a:\pctcp\protman.sys /i:a:\pctcp`
: Loads the Microsoft protocol manager, the /i option tells it where it can find the PROTOCOL.INI file.

`device=a:\pctcp\elnkii.dos`
: Loads the 3COM Etherlink II driver (use elnk3.dos for the Etherlink III).

`device=a:\pctcp\dis_pkt.gup`
: Loads the NDIS to packet driver converter. This driver allows the PC/TCP generic kernel, PC-210, to communicate with a specific NDIS device driver (in this case, elnkii.dos).

`files=10`
: Sets the maximum number of file handles. 10 should be more than sufficient for this application.

`buffers=20`
: Sets the number of file buffers. Disk performance is improved by adding more buffers, but each buffer occupies 512 bytes of memory. 20 buffers therefore equal 10 KBytes of memory, so if memory is tight, this setting can be lowered to a minimum of 3, particularly since the disk is not used during normal operations.

AUTOEXEC.BAT

Create an AUTOEXEC.BAT file as follows:

```
@echo off
prompt $p$g
set TZ=EST5EDT
set zyid=1
set ZYAZ=256
set ZYEL=260
set PCTCP=a:\pctcp\pctcp.ini
path a:\;a:\pctcp
rem The following lines load the network kernel
netbind
ethdrv
rem rload ftp gets remote.exe from a network server
rload host=bumda user=laser passwd=<password> hostdir=/laser/zy1
remote=remote.exe local=remote.exe
remote
```

The lines in the AUTOEXEC.BAT file have the following meanings:

@echo off
Prevents the commands in the AUTOEXEC.BAT file from being echoed to the display (optional)

prompt pg
Sets the command prompt to 'A:"' (optional)

set TZ=EST5EDT
Sets the TZ environment variable. This variable is needed by the ZY program and the PC/TCP clockset command to properly set the system clock.

set ZYID=1
Sets the ZYID environment variable. This variable is needed by the ZY program to know which ZY it is. In this case, it is ZY1

set ZYAZ=256
Sets the ZYAZ environment variable. This variable is used by the ZY program to determine the I/O address (decimal) of the azimuth axis of the 5328 motion controller board. Here it is set to 256 decimal (100 Hex).

set ZYEL=260
Sets the ZYAZ environment variable. This variable is used by the ZY program to determine the I/O address (decimal) of the elevation axis of the 5328 motion controller board. Here it is set to 260 decimal (104 Hex).

set PCTCP=a:\pctcp\pctcp.ini
Sets the PCTCP environment variable. This variable is used by the PC/TCP kernel to find it's initialization file, PCTCP.INI.

path a:\;a:\pctcp
Sets the execution path.

netbind
Calls NETBIND.EXE. NETBIND.EXE binds the NDIS device driver to the PC/TCP conversion module. This must happen before the PC/TCP kernel is called.

ethdrv
Calls ETHDRV.EXE, the PC/TCP generic kernel.

rload host=bunda user=laser passwd=<password> hostdir=/laser/zy1 remote=remote.exe local=remote.ex

Calls RLOAD.EXE. RLOAD.EXE is a command line FTP client that does a binary get of the ZY program file REMOTE.EXE from the host bunda, logging in as user laser. When rload terminates, REMOTE.EXE should be on the disk, waiting to be run.

remote
Calls the ZY program, REMOTE.EXE. See section 3.1 for more details. The ZY system should be ready to go after this command.

FTP PC/TCP

Create the subdirectory **A:\PCTCP** for the FTP PC/TCP files that will be required. Copy as a minimum the following files from the PC/TCP distribution diskettes to this directory:

elnkii.dos (or elnk3.dos) -- NDIS device driver
dis_pkt.gup -- NDIS to packet driver converter

```
ethdrv.exe -- the PC/TCP kernel
netbind.exe -- Binds the kernel to the packet driver converter
protman.sys -- MS Protocol manager
rload.exe -- A small FTP client, used to load ZY program
ping.exe -- used to verify network connections
protocol.ini* -- the protocol manager's initialization file
pctcp.ini* -- the kernel's initialization file
```

NOTE: Files marked with <*> may have to be created, and will certainly have to be modified to accomodate differences between each ZY, such as network IP address, and possible differences in hardware (Etherlink II vs Etherlink III, for instance). Please refer to the PC/TCP and 3COM documentation to see how this is done. Sample PCTCP.INI and PROTOCOL.INI files also can be found in the 'laser' account.

Once this disk has been created, a mirror of it should be kept in the 'laser' account (in the subdirectory 'laser/remotes/syxxx' where xxx represents the ZY number in question. This will make it easy to recover in the event of a floppy disk failure.

2.2.2 ZY Directory

After all of this is complete, the ZY directory should look like this.

```
CONFIG.SYS
AUTOEXEC.BAT
CUBES.INI
ZY.INI
CUBES.BAK
ZY.BAK
REMOTE.LOG
REMOTE.EXE
\PCTCP
     ELNKII.DOS
     DIS_PKT.GUP
     ETHDRV.EXE
     NETBIND.EXE
     PROTMAN.SYS
     RLOAD.EXE
     PING.EXE
     PROTOCOL.INI
     PCTCP.INI
```

2.3 Network Files

A user account named 'laser' is needed by the ZY systems to properly boot and initialize themselves. See your Systems Administrator if this has not already been set up. The following subdirectories must be present for the ZY system to boot and initialize:

/laser/zy<N>
: Location of ZY executable REMOTE.EXE (where N stands for the number of the ZY system in question. For ZY 1, that would be /laser/zy1).

/laser/init
: Location of the ZY initialization files

/laser/remotes
Location of the ZY diskette mirror archives.

Note that these paths are relative to the 'laser' home directory. Currently, the account is established on host cestus, and the full path for **/laser/init** is **/cestus1/laser/laser/init**. Such a path specification must not be used either in the REMOTE.EXE program or as a command line to RLOAD.EXE, as the location of the 'laser' account cannot be guaranteed.

2.3.1 Laser Directory

For a three laser system, the network laser directory should look like this.

```
~/laser/zy1
     remote.exe
~/laser/zy2
     remote.exe
~/laser/zy3
     remote.exe
~/laser/init
     cubes.ini
     zy001.ini
     zy002.ini
     zy003.ini
~/laser/remotes/zy001
     CONFIG.SYS
     AUTOEXEC.BAT
     CUBES.INI
     ZY.INI
     CUBES.BAK
     ZY.BAK
     REMOTE.LOG
     REMOTE.EXE
          \PCTCP
               ELNKII.DOS
               DIS_PKT.GUP
               ETHDRV.EXE
               NETBIND.EXE
               PROTMAN.SYS
               RLOAD.EXE
               PING.EXE
               PROTOCOL.INI
               PCTCP.INI
~/laser/remotes/zy002
     CONFIG.SYS
     AUTOEXEC.BAT
     CUBES.INI
     ZY.INI
     CUBES.BAK
     ZY.BAK
     REMOTE.LOG
     REMOTE.EXE
          \PCTCP
```

```
                ELNKII.DOS
                DIS_PKT.GUP
                ETHDRV.EXE
                NETBIND.EXE
                PROTMAN.SYS
                RLOAD.EXE
                PING.EXE
                PROTOCOL.INI
                PCTCP.INI
~/laser/remotes/zy003
     CONFIG.SYS
     AUTOEXEC.BAT
     CUBES.INI
     ZY.INI
     CUBES.BAK
     ZY.BAK
     REMOTE.LOG
     REMOTE.EXE
          \PCTCP
                ELNKII.DOS
                DIS_PKT.GUP
                ETHDRV.EXE
                NETBIND.EXE
                PROTMAN.SYS
                RLOAD.EXE
                PING.EXE
                PROTOCOL.INI
                PCTCP.INI
```

Chapter 3

Operation

3.1 Booting the ZY

As described in section 2.2.1, the ZY computers boot from a local 1.44 MByte floppy. DOS and the network software reside on this floppy, as well as a simple FTP client program, RLOAD.EXE, whose sole purpose is to retrieve the ZY executable. RLOAD.EXE is called from the AUTOEXEC.BAT file after the network kernel ETHDRV.EXE has been called. RLOAD.EXE must be called with the following parameters, all separated by a space:

```
host=<hostname>
user=<username>
passwd=<password>
hostdir=<hostpath>
remote=<hostfilename>
local=<localfilename>
```

where:

<hostname> is the name of the host that is running the ftp server

<username> is the name of the account, in this case 'laser'

<password> is the password to the 'laser' account

<hostpath> is the path to the 'remote.exe' file

<hostfilename> is the name of the ZY executable on the UNIX host, in this case 'remote.exe', in lowercase.

<localfilename> is the name of the ZY executable on the local MS-DOS ZY host, in this case 'REMOTE.EXE', case is not important here.

After RLOAD.EXE successfully terminates, REMOTE.EXE is executed.

3.2 Connecting to ZY

Once the ZY is booted up and running, it sets up a listening TCP socket. To connect to this listening socket, the client program must make a TCP connection to host zy<n> (where n is the ZY number) at port 5240. When the connection is made, the client program may wish to verify the connection by sending a status request command (STS or STW). If all is well, the ZY will send back a status

string (STS) or a status word (STW). While the ZY and the client program are connected, the ZY will send a newline character to the client every 60 seconds to make sure that the client has not been disconnected. If it has, the ZY will close the connection and resume listening. The client program should therefore interpret a single newline character as a "null" command and do nothing.

Note that it is possible to connect to the ZY with a standard telnet program, as in the following example for zy1 and the Sun OS telnet:

```
telnet zy1 5240
```

This is useful for trouble-shooting purposes or to familiarise oneself with the command syntax of the ZY. Set the telnet program to local echo and send on end of line. Some telnet programs will correctly set themselves to this basic setup because the ZY program is not really a telnet server and therefore the telnet negotiation will fail.

3.3 Using Commands

There are two broad classes of commands in the ZY command set: low level and high level (or cube based). Low level commands allow the client program to directly control the various sub-systems or system parameters of the ZY such as the servo systems, the A/D system, system status, etc. High level commands use one or more sub-systems to perform the tasks that the ZY was designed to accomplish.

3.3.1 Syntax

Commands are issued in a 7-bit ASCII stream, and consists of the command itself followed by 0 or more parameters, separated by a comma (,) and terminated by a newline character. The ZY is case insensitive. In the following discussion, items in angle brackets (< >) are mandatory, and items in square ([ ]) brackets are optional.

The general command syntax is:

```
COMMAND [param, param...]  newline
```

When the ZY has processed the command, it replies as follows:

```
COMMAND <1>, [param, param...]  newline
```

if the command succeeded, or

```
COMMAND <0>, [error message string] newline
```

if the command failed. In case of success, the parameter list that the ZY sends back is the same that was sent to it.

The ZY buffers the commands sent to it, so that the client program need not wait for a reply before sending the next command. At this point, an example may prove helpful; the following command sets the absolute acceleration of the elevation axis to 10 000:

```
ABA 1, 10000
```

Note that a space is required between the command and the first parameter. The ZY will eventually reply with:

```
ABA 1, 1, 10000
```

if the operation succeeded, or

```
ABA 0, 1, error loading acceleration
```

if the operation failed.

Most commands that set new values can be used without the value parameter to request the currently set value. In the event that our example command succeeded, then sending

```
ABA 1
```

will request from the ZY the currently set acceleration value for axis 1 (the elevation axis) and the ZY will reply with:

```
ABA 1, 1, 10000
```

As mentioned above, there are two types of commands in the ZY command set: Low level commands, used to directly manipulate a subsystem, and higher level commands (called 'cube based commands' because they take a cube name or number as a parameter), which use the subsystems at their disposal in the course of performing their function. For instance, to move the laser beam to cube ZG11, one could do the following with low level servo commands:

```
ABP 0, 29000      ; give azimuth a destination of 29000
ABP 1, 65700      ; give elevation a destination of 65700
STT 0             ; start the azimuth moving
STT 1             ; start the elevation moving
WAI 0             ; wait for azimuth to complete move
WAI 1             ; wait for elevation to complete move
```

where 29000 and 65700 are arbitrary encoder coordinates that woud aim the laser at the desired cube. Using a cube based command to do the same thing would look like this:

```
CIL ZG11          ; aim laser at cube ZG11
CWT ZG11          ; wait for operation to complete
```

This is much more sophisticated since the ZY can calculate the needed azimuth and elevation encoder coordinates from the actual three dimensional coordinate of the cube's location. Cube based commands cannot be used if cube objects have not been created. They will fail if the cube parameter passed to them is incorrect (cube does not exist, for instance). The ones that use the servo sub-system will fail if the servos have not been homed.

3.4 Initializing the ZY

After the client program has verified the connection, it is ready to make measurements. At this point, though not absolutely nescessary, the client program should initialise the ZY program, by sending the `INITZY` command. When the ZY receives this command, it will retrieve (via binary ftp) two initialisation files from the 'laser' account: CUBES.INI and ZYxxx.INI (where xxx is the ZY number, a 3 digit number whith leading zeroes if nescessary; for example: ZY001.INI for ZY1).

CUBES.INI contain commands that are used by every ZY, and ZYxxx.INI contains commands that are specific to a particular ZY, so that there will be one ZYxxx.INI for each ZY in the system. These files are located in the directory /laser/init. The ZY program saves these localy to A:CUBES.INI and A:ZY.INI, renaming any previous copies to A:CUBES.BAK and A:ZY.BAK.

The ZY then executes the commands in CUBES.INI and then ZY.INI, sending it's responses to these commands back to the client program, as if the client program had sent these commands itself. This will allow the client program to initialise any of it's data structures as nescessary.

Finally, when all the commands contained in the two initialisation files have all been executed, the ZY will acknowledge the INITZY command by sending the string 'INITZY 1' back to the client, if the ftp get was successful, or 'INITZY 0, <error msg>' if the ftp failed. In the case where the ftp get failed, the ZY program will have restored CUBES.INI and ZY.INI from the backup files CUBES.BAK and ZY.BAK, and executed the commands from these older initialisation files.

It is the client program's responsibility to decide how to proceed in the event of an INITZY failure. Note that if this is the first time that the ZY was initialised, there may be no old copies of CUBES.INI and ZY.INI, and that even if old copies exist, the information contained within them may be out of date, so it is highly recommended that the client program terminate in the event of an INITZY failure and that the cause of the problem be determined.

3.4.1 CUBES.INI example

A typical CUBES.INI file would look like this.

```
; CUBES.INI
; Device independent cube information:
;  - Total number of cubes
;  - Name and 3 dimensional coordinates of each cube
;  - Number and order of cube scan
;
; allocate enough memory for 17 cubes
INI 17
;
; initialize each cube:
; coo <index>, <name>, X, Y, Z, AZ, EL
;
coo 0, ZRG, 0, 0, 0, 0, 0
coo 1, ZBG, -88148.4340, -203303.0630,  -328.2700, 0, 0
coo 2, ZG11, -78876.7230, -208044.3490,  1786.1280, 0, 0
coo 3, ZG12, -80502.6480, -206734.4100,  1786.1280, 0, 0
coo 4, ZG13, -82135.3700, -205420.8440,  1792.2240, 0, 0
coo 5, ZG21, -77408.1360, -206207.9290,   822.9600, 0, 0
coo 6, ZG22, -79034.8530, -204903.7200,   822.9600, 0, 0
coo 7, ZG23, -80669.2520, -203592.1960,   826.0080, 0, 0
coo 8, ZG31, -75921.3520, -204369.4060,  -143.2560, 0, 0
coo 9, ZG32, -77559.1640, -203064.7090,  -137.1600, 0, 0
coo 10, ZG33, -79199.6280, -201752.1490,  -143.2560, 0, 0
coo 11, ZBG1, -122982.8380, -258786.3250,  -300.7640, 0, 0
coo 12, ZBG2, -88148.4340, -203303.0630,  -328.2700, 0, 0
coo 13, ZBG3, -75755.7240, -213554.4010,  -328.2700, 0, 0
coo 14, ZBG4, -40683.7900, -157995.2440,  -309.9820, 0, 0
coo 15, ZBG5, -51539.7600, -119544.2970,  -309.9820, 0, 0
coo 16, ZBG6, -2510.0280, -158643.0360,  -331.2260, 0, 0
;
; number of retros to scan:
;
```

```
NUM 11
;
;
; Scan order: The first number is the scan list starting position,
; the rest are the retro number list.
;
ord 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Explanation

`INI 17`
Sets up 17 cube objects in the ZY computer's memory

`COO 0, ZRG, 0, 0, 0, 0, 0`

...

`COO 16, ZBG6, -2510.0280, -158643.0360, -331.2260, 0, 0`
These commands initialise each cube with their index number, name, XYZ coordinate, and their AZ EL coordinate. The AZ and EL coordinates where once nescessary to tell the servo system where to point, but are now obsolete. They are kept for backwards compatibility, and may be set to 0.

`NUM 11`
Sets the size of the scan list. The ZY uses a scan list to determine which cubes to measure if the SCN command is given.

`ORD 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10`
Fills the scan list with the cube index numbers of the cubes to scan.

3.4.2 ZY001.INI example

A typical ZY001.INI file would look like the following.

```
; ZY001.INI
;
; last changed: 08/24/93 R Creager
;
;
; Reference Retro Azimuth \& Elevation
;
AZM ZRG, -23538
ELV ZRG, -29002
;
;
; Laser position constants: These are used to calculate the
; laser system's position and orientation in 3 dimentional
; space.
;
BX -21945.6
BY -134696.363
BZ 31592.52
```

```
AZO 0
ELO 20735
XO1 -11727.8
XO2 -0.98705
XO3 0.022797
YO1 -25110.2
YO2 0.003107
YO3 0.99838
INVC
;
;
; A/D info:
;
IFF 1000    ; 1000 kHz reference frequency
CYC 128     ; 128 cycles per measurement
SFQ 64      ; 64 samples per cycle
;
;
; Servo (LM628) info, azimuth:
;
ABV 0, 15000000   ;LM628 velocity value
ABA 0, 10000      ;LM628 acceleration value
FKP 0, 200        ;LM628 proportional (gain) value
FKI 0, 300        ;LM628 integration term
FKD 0, 3500       ;LM628 differetiation term
FIL 0, 20         ;LM628 integration limit
FSI 0, 1          ;LM628 differential sampling rate
ERL 0, 5000       ;LM628 position error limit
;
; Servo (LM628) info, elevation:
;
ABV 1, 15000000
ABA 1, 20000
FKP 1, 50
FKI 1, 50
FKD 1, 600
FIL 1, 100
FSI 1, 2
ERL 1, 5000
;
;
; Servo Wait parameters, azimuth
;
WMD 0, 0
WCNT 0, 5
WTOL 0, 2
WTMO 0, 18
;
;
; Servo Wait parameters, elevation
;
```

```
WND 1, 0
WCNT 1, 5
WTOL 1, 2
WTNO 1, 18
```

Explanation

AZM ZRG, -23538

ELV ZRG, -29002

Sets the asimuth and elevation encoder coordinates for the reference cube (ZRG). The reference cube is the only cube that must have AZ and EL values set. The reference cube is used to compensate for electronic drift in the ZY instrument and is mounted directly on the instrument. Since it moves and tilts with the instrument, it can always be found with encoder coordinates and therefore no coordinate transformation is required, or indeed possible.

BX -21945.6

BY -134696.363

BZ 31592.52

AZO 0

ELO 20735

XO1 -11727.8

XO2 -0.98705

XO3 0.022797

YO1 -25110.2

YO2 0.003107

YO3 0.99838

INVC

These commands tell the laser where it is located in 3 dimensional space, and how it is oriented. The INVC command also tells each cube that the laser's orientation and location may have changed, and that new encoder coordinates must be calculated.

IFF 1000

1000 KHs reference frequency

CYC 128

128 cycles per measurement

SFQ 64

64 samples per cycle. These commands initialize the A/D sub-system.

ABV 0, 15000000

LM628 velocity value

ABA 0, 10000

LM628 acceleration value

FKP 0, 200
LM628 proportional (gain) value

FKI 0, 300
LM628 integration term

FKD 0, 3500
LM628 differetiation term

FIL 0, 20
LM628 integration limit

FSI 0, 1
LM628 differential sampling rate

ERL 0, 5000
LM628 position error limit These commands load the servo controller's trajectory values and PID (proportional, integral, derivative) filter. This sequence initialises the azimuth (0) axis, and should be repeated for the elevation (1) axis as well.

NOTE: It is essential that the velocity value be loaded first. The National Semiconductor LM628 motion controller used in the ZY requires that the acceleration value must not exceed the velocity value, or a command error will result. If the acceleration value is loaded first, and the velocity value is still 0 (power up default), then the operation will result in a command error, meaning that the acceleration value will not have been loaded. This will result in a servo system that refuses to move.

WND 0, 0

WCNT 0, 5

WTOL 0, 2

WTMO 0, 18
These commands determine how the servo system will wait until a move is complete. These commands are for axis 0. They must also be sent for axis 1.

3.5 Servo Operation

3.5.1 Preparing the Servo System for Operation

Before the servo system can move the servo motors, the servo controller's PID (Proportional, Integral, and Derivative) filter values and trajectory (acceleration and maximum velocity) values must be loaded, as explained in section 3.4. To date, the values used have been determined by trial and error, and because each mechanism is different in some way (different friction etc.) these values are different from one ZY to the next.

3.5.2 Finding Home Position

The encoders used to control the servo motors on the ZY system are incremental encoders. When the system is first powered on, the position they happen to be in becomes the home position (encoder coordinate 0). This is a purely arbitrary position.

Before the encoders can be used to point the laser at any absolute coordinate, the servo systems must first undergo a procedure that locks in their true home positions. This is accomplished with the 'FHM' command for each axis. FHM will rotate the specified axis until a special index pulse is

noted. The servo controller's position register is then altered so that it reads 0 at the location of that index pulse.

Since the location of the index pulse is physically fixed, the encoder's coordinates can now be considered absolute. This procedure needs to be performed only once after a system startup. As long as the ZY program is running, the servo controllers will remember their respective home position. The "homed" condition of the servo system can be determined by reading the system status word, so that a client program need not perform this procedure unnecessarily (see STS, STW).

NOTE: Some commands will fail if this procedure was not performed or if it failed.

3.5.3 Using the Servos

With the PID and trajectory values loaded, the servo motors can be moved to any point in their range with the low level servo commands. Moving an axis involves first loading the destination, then issuing a start command. For example

```
ABP 0, 3000
STT 0
WAI 0
```

moves axis 0 to encoder coordinate 3 000. WAI blocks until the move completes. The use of WAI is optional. It should be used if synchronisation with the move is nescessary.

3.6 Data Acquisition

3.6.1 Data Acquisition Sub-System

The ZY laser ranging system uses a 780 nanometer (nm) wavelength laser modulated at 1500 MHz to determine the exact distance to a retro-reflector in the field. The reflected signal is received by a PIN silicon photodiode, mixed, digitised by the 16 bit 100 KHz A/D system, and processed to extract the phase.

Ideally, the phase of the received 1500 MHz signal would then be compared against a 1500 MHz reference signal to determine the phase difference, and thus the exact location of the retro-reflector. However, The A/D sub-system is incapable of sampling a signal of this frequency. To get around this problem, the returning 1500 MHz signal is first mixed with a (1500 MHz + IF reference) signal to produce an IF signal of much lower frequency, which can then be easily sampled. The phase of the IF signal is then compared to the IF reference.

To make a measurement, the system samples a number of IF cycles at a predetermined sampling rate. The total number of samples taken is then the product of the number of cycles integrated and the sampling rate, expressed in samples per cycle. The number of cycles to measure, the sampling rate, and the frequency of the IF reference are all programmable.

3.6.2 Direct Memory Access (DMA)

In order to minimise the workload of the computer's CPU, the A/D system is configured to use DMA services to store the acquired data. An AT class computer, such as the computer used in the ZY system, has 2 DMA controllers, one providing 8 bit DMA services on DMA channels 0-3, and the other providing 16 bit services on channels 5-7. Both will allow a maximum of 65 536 words. The ZY's A/D hardware is configured to use 16 bit DMA channel 5, and the software sets aside a 128 KByte DMA buffer to store the samples.

To perform a measurement, the appropriate DMA controller is preloaded with the total number of samples desired, and is then given a software trigger. The A/D system then synchronises with the IF reference, and takes samples until the DMA terminal count is reached. As an example, consider

a measurement made with the ZY's default cycle and samples-per-cycle values. The measurement will integrate over 128 cycles, with 64 samples per cycle, and the number of 16 bit words in the DMA buffer will total 8192. The DMA buffer can now be thought of either as an array of 8192 individual 16 bit samples, or as an array of 128 structures, each structure representing a complete set of 64 samples of an IF cycle. The phase calculation algorithm treats the buffer as the latter.

3.6.3 Using A/D commands

The commands CYC (number of cycles sampled), IFF (IF reference), and SFQ (samples per cycle) are used to configure the A/D system to make measurements. When using the A/D commands CYC, IFF, and SFQ, it is important to note that the hardware imposes some limits. The DMA restriction of 65 536 samples has been noted above. The product of the CYC and SEQ values should not exceed this limit.

The other important limitation is imposed by the A/D's maximum sampling frequency of 100 KHz. In this case, the product of the SEQ and IFF values should not exceed this maximum sampling frequency. In either case, the ZY will not allow an illegal condition: the offending command will return with an error.

Once the A/D system has been properly configured, the client program can use either low level commands or high level commands to make a measurement. The following examples show how to use the low level commands to perform measurements:

To integrate a number of cycles and calculate a phase and amplitude:

```
TRG     ; trigger the measurement
MPC     ; calculate phase and magnitude from data in DMA buffer
RAD     ; return phase, in radians
MAG     ; return magnitude, in volts
```

The ZY responds with:

```
TRG 1
MPC 1
RAD 1, 3.6657776
MAG 1, 2.67
```

To calculate a sequence of phases and amplitudes, each one corresponding to a cycle structure in the DMA buffer:

```
TRG     ; trigger the measurement
SEQ     ; calculate and return the sequence data
```

The ZY responds with:

```
TRG 1
SEQ 1, 0, 3.534, 4.23489882, 0, 0
SEQ 1, 1, 3.499, 4.23398948, 0, 0
...
SEQ 1, n, 3.529, 4.23459893, 0, 0
```

where n is (number-of-cycles - 1). The 2 zeroes at the end of each line are place holders for an undocumented experimental feature and may be safely ignored.

To retrieve a range of individual samples from the DMA buffer:

```
TRG;      ; trigger the measurement
DAT 0, 9 ; request samples 0 through 9
```

The ZY responds with:

```
TRG 1
DAT 1, 0, -22094
DAT 1, 1, -20138
...
DAT 1, 9, 30982
```

High level commands are very similar to their low level counterparts, except that the data in the DMA buffer is associated with a cube, and the results of the phase and amplitude are stored in that cube's data structure, and can be used to return actual distance measurements (with low level commands, the results are not stored anywhere outside the phase system's structures, and are overwritten by the next calculation). The following example shows how to use high level commands to make a distance measurement:

```
CIL ZG11   ; move laser to cube ZG11
CWT ZG11   ; wait for laser to get there...
CTR ZG11   ; trigger measurement.  DMA data belongs to ZG11
CLC ZG11   ; calculate phase and amplitude for ZG11
AMP ZG11   ; get calculated amplitude for ZG11 (in volts)
PHI ZG11   ; get calculated phase for ZG11 (in radians)
DST ZG11   ; get actual distance to ZG11 (in mm)
```

The ZY responds with:

```
CIL 1, 4   ; in this case, 4 is the index number of ZG11
CWT 1, 4
CTR 1, 4
CLC 1, 4
AMP 1, 4, 3.45
PHI 1, 4, 3.14159265
DST 1, 4, 100454.345
```

There are no high level equivalents to the SEQ and DAT commands. There is, however, another high level data acquisition command that merits mention: SCN. SCN causes the ZY to measure in turn each cube listed in the scan list. The NUM and ORD commands are used to maintain this list (see NUM, ORD, SCN). If a large number need to be measured, SCN is preferable to repeating the example above for each cube, for the following reasons:

1. It is optimised for speed (see following section).
2. It requires a single command from the client. This is particularly important in a multiple ZY system, as each can scan the list of cubes concurrently, without the need of multiple commands from the client program, which necessarily would have to send them in sequence to each ZY.

3.6.4 Data Acquisition Timings

Several factors will dictate how long a measurement will take to perform. For a simple measurement, where the laser is already pointed at a cube, the measurement consists of two parts: data acquisition and data processing. The SCN scan command is more complex. In this case, the servo translation times must also be taken in account.

Simple Measurement Times

The time taken by the data acquisition portion is the product of the number of cycles integrated and the period of each cycle. For example, using the system defaults of 128 cycles and 1 ms (1 kHz IF) respectively, this would work out to an integration time of 128 ms, or 1/8th of a second.

The data processing time is influenced by two factors: The number of samples processed, and the CPU and FPU speed. The more samples there are, the longer the phase calculation will require.

The number of samples affect the calculation in another way: there are two distinct phase calculation routines, one written in Intel 286/287 assembly language, and one written in a high level language (C++). Due to the segmented architecture of the Intel 286, the assembly language phase routine (as it is now written) can be used with a maximum of 32768 samples. Because the system is capable of acquiring up to 65 536 samples, the system will automatically switch to the high level language routine when the number of samples exceeds 32768, and the calculations will take approximately 5 times longer to complete.

Again using the system defaults as an example, a 286/287 AT running at 16 MHz can reduce 8192 samples to phase and amplitude values in 33 ms.

SCN Timings

The SCN routine takes advantage of the fact that the National LM628 servo controller and the Intel 8237 DMA controller both operate independently of the main CPU to reduce the measurement overhead. It exploits the independence of the LM628 by commanding the servo controllers to move the laser to the next cube in the list while the CPU is calculating the phase and amplitude of the current cube on the list. It then communicates these results to the client program during data acquisition for the next cube. If the servos are optimally tuned, this technique removes from consideration the time it takes to translate the servos from one cube to the next, and the time needed to communicate the results of the previous measurement to the client program.

3.7 Trouble-shooting

When troubleshooting the ZY, it may be extremely useful to make a TELNET connection to the ZY to try to duplicate the condition by entering commands manualy (see section 3.2). This provides immediate feedback and also separates the client program from the ZY (the bug may after all be in the client program).

The ZY implements a watchdog timer. This timer will reset the system if processing comes to a halt for any reason. The watch dog timer keeps a log file in the root directory called REMOTE.LOG. This file logs the time and date of the reset, an index to the command that was executing during reset, and an optional source code line number. The best way to quickly view REMOTE.LOG is to use the command "RDF REMOTE.LOG" in a TELNET session (see RDF). The file will scroll by the telnet screen. If this file is long, it may be difficult to view the top. The most recent resets are at the bottom of the file.

3.7.1 Communications Problems

Problem: ZY is up and running, but connection is refused, or connection is read-only.

Possible Cause: Some other program may already be connected to the ZY.

Remedial Action: Wait for ZY to become free or disconnect the other program.

Problem: ZY is up and running, but host cannot be found.

Possible Cause: The network software on the ZY may not be properly configured.

Remedial Action: Check the connection with the 'ping' command from another host. If this fails, verify the configuration of the network software on the ZY.

3.7.2 Servo Problems

Problem: Servo system acts normal, but servos do not move.

Possible Cause: Either the acceleration value or the velocity value is currently set to 0. This will cause the LM628 to signal a completed move, but not move the servos. This problem typically arises when the acceleration value is loaded before the velocity value during initialization. When the system is started, the LM628 comes up with both the velocity and acceleration set to 0. By loading the acceleration first, the requirement that the acceleration value be less than the velocity value is violated and the acceleration is left at it's current value, 0.

Remedial Action: Place the ABV command before the ABA command during initialization.

Problem: Servo move takes a long time to complete, or do not complete at all.

Possible Cause: The servo system is not properly tuned, causing the servo motors to oscillate about their destination. This problem usually occurs during operations that require the servo to settle on target, such as FHM, CWT, WAI, VHM or SCN.

Remedial Action: Tune the servos.

Problem: Some high level commands fail with the message "axis n not homed".

Possible Cause: The servo axis 'n' is not homed. Coordinates used on this axis will be invalid, so operations that require absolute servo coordinates will fail.

Remedial Action: Execute FHM for that axis

3.7.3 Data Acquisition Problems

Problem: Measured amplitude is zero for all cubes.

Possible Cause: The oscillator may not locked to the 100 MHz reference or to the IF reference. This is a safe bet if the amplitude is exactly 0.00 for all cubes, including the reference cube (ZRG).

Remedial Action: Ensure that the 100 MHz reference and the IF reference are getting to the oscillator. If so, trouble-shoot oscillator.

Problem: Measured amplitude is low for some cubes.

Possible Cause: The servos are not pointing to a corner cube. The corner cube may be covered (by dew or frost, for instance) or an obstacle may be in the optical path.

Remedial Action: Check coordinates of the cube under test. Check for any obstructions in the optical path.

Chapter 4

List of Commands by Function

4.1 Instrument Control

These are general instrument control/status commands.

```
BYE     - Disconnect from ZY
GTI     - Get ZY Time
INI     - Create cube objets
INITZY  - Initialize the ZY program
QQQ     - Terminate the ZY program
RDF     - Read a file on the ZY's disk
RST     - Reset the ZY computer
STI     - Set ZY Time
STS     - Return ZY status string
STW     - Return ZY status word (2 byte)
VER     - Returns the ZY version number
```

4.2 Data Acquisition

These commands allow direct control of the data acquisition sub-system.

```
CYC     - Sets/returns the number of cycles of the IF signal to sample
DAT     - Returns one or more samples from memory
IFF     - Sets the frequency of the IF signal.
MAG     - Returns the last calculated magnitude
MPC     - Commands the Data Acquisition sub-system to calculate
          a phase and magnitude from data stored in memory
RAD     - Returns the last calculated phase, in radians
SEQ     - Returns a sequence of magnitude and phase values, each
          belonging to an individual cycle from a string of cycles
          stored in memory
SPQ     - Sets/returns the number of samples per cycle
TRG     - Commands Data Acquisition system to begin acquire new data
```

4.3 Servo

The servo sub-system controls 2 axes of movement. The axis number must be specified when using each one of the following commands.

```
ABA    - Sets/returns the absolute acceleration
ABP    - Sets a target position for a subsequent move
ABV    - Sets/returns the absolute velocity
ACP    - Returns the current encoder position
AXS    - Returns the axis status word
CLE    - Clears the axis error flag
DSP    - Returns the desired encoder position
ERL    - Sets/returns the position error limit
FHM    - Finds home
FIL    - Sets/returns PID filter integration limit
FKD    - Sets/returns PID filter derivative constant
FKI    - Sets/returns PID filter integration constant
FKP    - Sets/returns PID filter proportional constant
FLT    - Sets/returns all PID filter constants
FSI    - Sets/returns PID filter derivative sampling interval
IDX    - Returns current index position
LIMIT  - Sets/returns 'software stops' for the servo motor
RDS    - Returns the integration sum
STT    - Starts the motor on a move
VHM    - Verify home position
WAI    - Wait for move to complete
WCNT   - Sets/returns the 'wait count' parameter
WTMO   - Sets/returns the 'wait time out' parameter
WTOL   - Sets/returns the 'wait tolerance' parameter
WMD    - Sets/returns the 'wait mode' parameter
```

4.4 Pointing

Most of these commands are used to initialize the ZY. They alter system parameters that allow a ZY to point it's laser to a 3 dimensional coordinate, instead of using raw encoder coordinates. The actual encoder coordinates are calculated as

$$az = \mathrm{X01} + \mathrm{X02}(\theta + \mathrm{AZ0}) + \mathrm{X03}(\phi + \mathrm{EL0}) \tag{4.1}$$

where θ and ϕ are defined on drawing D35420M051, and X01, X02, and X03 are constants measured in the calibration lab. This equation will be modified to include monument constants in the final form.

```
AZO    - Sets/returns azimuth offset
BX     - Sets/returns base X coordinate of ZY
BY     - Sets/returns base Y coordinate of ZY
BZ     - Sets/returns base Z coordinate of ZY
ELO    - Sets/returns elevation offset
INVC   - Invalidates current coordinate az/el coordinates for each cube
XO1    - Sets/returns X 1st order constant
XO2    - Sets/returns X 2nd order constant
XO3    - Sets/returns X 3d order constant
YO1    - Sets/returns Y 1st order constant
YO2    - Sets/returns Y 2nd order constant
YO3    - Sets/returns Y 3d order constant
```

4.5 High level, or Cube Commands

Each one of these commands takes a cube name or number as a parameter. The value or action specified belongs to that cube.

AMP - Returns last calculated amplitude
AZM - Sets/returns azimuth coordinate
CIL - Illuminate cube with laser
COO - Sets/returns cube index, name and coordinates
CTR - Triggers data acquisition
CWT - Blocks until CIL has completed
CX - Sets/returns X coordinate
CY - Sets/returns Y coordinate
CZ - Sets/returns Z coordinate
DST - Returns last calculated distance
ELV - Sets/returns elevation coordinate
NUM - Sets/returns the number of cubes to scan
ORD - Sets/returns the order of a scan
PHI - Returns last calculated phase
SCN - Scans all the cubes in the scan list

Chapter 5

Detailed Description of Commands

ABA

Mnemonic	ABsolute Acceleration
Function	Sets absolute acceleration for LM628
Syntax	ABA <n>[, a] n: axis value (0: asimuth, or 1: elevation) a: acceleration value 0 to $(2^{30}) - 1$ If <a> is ommitted, current value is returned.
Returns	On Success: axis number, and new or current acceleration value. On Failure: axis number and an error message.
Remarks	The acceleration value must not exceed the velocity value. It will take effect the next time the servos are moved. For an explanation of the units used for the acceleration, see pg. 14 of the National Semiconductor data sheet for the LM628.
Example	send: ABA 1, 20000 receive: ABA 1, 1, 2000 send: ABA 0 receive: ABA 1, 0, 1000
See Also	ABV, STT

ABP

Mnemonic	ABsolute Position
Function	Sets target destination in absolute encoder counts.

Syntax ABP <n, p>
n: axis value (0: asimuth, or 1: elevation)
p: position in encoder counts $-(2^{30})$ to $(2^{30}) - 1$
If this value is not present, ABP returns an error. Current position can be obtained using ACP for actual position or DSP for desired position.

Returns On Success: axis number and the new position.
On Failure: axis number and an error message.

Remarks It is important to keep in mind that the units used are encoder counts, and are therefore dependent on the encoder used. In order to complete the move, STT <n> must be issued.

Example send: ABP 0, 15000
receive: ABP 1, 0, 1500

See Also ACP, STT

ABV

Mnemonic ABsolute Velocity

Function Sets the absolute velocity to be used in all subsequent moves.

Syntax ABV <n>[, v]
n: axis value (0:asimuth or 1:elevation)
v: velocity value 0 to $(2^{30}) - 1$
If <v> is ommitted, current value is returned.

Returns On Success: axis value and the new or current velocity value
On Failure: error message.

Remarks The velocity value must equal or exceed the acceleration value. It will take effect the next time the servos are moved. For an explanation of the units used for the velocity, see pg. 14 of the National Semiconductor data sheet for the LM628.

Example send: ABV 1, 2000000
receive: ABV 1, 1, 2000000

send: ABV 1
receive: ABV 1, 1, 2000000

See Also ABA, STT

ACP

Mnemonic ACtual Position

Function Returns the current encoder count for the specified axis.

Syntax	ACP <n> n: axis number (0:asimuth or 1:elevation)
Returns	On Success: axis value and current encoder count On Failure: error message.
Remarks	The returned value will be in the range $-(2^{30})$ to $(2^{30})-1$
Example	send: ACP 0 receive: ACP 1, 0, -29885
See Also	DSP

AMP

Mnemonic	AMPlitude
Function	Requests the most recent signal magnitude for specified cube.
Syntax	AMP <n> n: cube number.
Returns	On Success: cube number and signal magnitude in volts. On Failure: cube number and error message.
Remarks	This is a cube based command, therefore INI must be called before this command can be successful. AMP will fail if cube<n> is not in range.
Example	send: AMP 5 receive: AMP 1, 5, 0.385
See Also	CIL, CLC, COO, CTR, INI, PHI

AXS

Mnemonic	AXis Signals register
Function	Requests the signal word for the specified axis controller
Syntax	AXS <n> n: axis number
Returns	On Success: axis number and LM628 status word (in hexadecimal). On Failure: error message.

Remarks The status word will be returned as a 16 bit hexadecimal number. The 16 bits have the following meaning:

Bit 15 : Host Interrupt
Bit 14 : Acceleration Loaded (But Not Updated)
Bit 13 : UDF Executed (But Filter Not Yet Updated)
Bit 12 : Forward Direction
Bit 11 : Velocity Mode
Bit 10 : On Target Bit 9 : Turn Off Upon Excessive Position Error
Bit 8 : Eight-Bit Output Mode
Bit 7 : Motor Off
Bit 6 : Breakpoint Reached
Bit 5 : Excessive Position Error
Bit 4 : Wraparound Occured
Bit 3 : Index Pulse Acquired
Bit 2 : Trajectory Complete
Bit 1 : Command Error
Bit 0 : Acquire Next Index

For a full explanation of these flags, see pg 15 of the National Semiconductor LM628 data sheet.

Example send: AXS 0
receive: AXS 1, 0, 0x0604

AZ0

Mnemonic AZimuth zero

Function Load a new azimuth offset value for the coordinate transformation module, or optionaly requests the old value, if parameter is ommitted.

Syntax AZ0 [value]
value: azimuth encoder count offset $-(2^{30})$ to $(2^{30})-1$

Returns On Success: new azimuth offset value, if one was specified, or the current value if not.
On Failure: This function should not fail.

Remarks This function, allong with it's companion functions, load coordinate transformation constants that allow the laser to correctly point to any 3 dimensional coordinate in an arbitrary coordinate system.

Example send: AZ0 23000
receive: AZ0 1, 23000

send: AZ0
receive: AZ0 1, 23000

See Also BX, BY, BZ, EL0, XO1, XO2, XO3, YO1, YO2, YO3, INVC

AZM

Mnemonic cube AZiMuth

Function Loads new azimuth encoder coordinate for the specified cube. If no value is specified, requests current value.

Syntax AZM <c>[, enc_val]
c: cube number or name
enc_val: new encoder value $-(2^{30})$ to $(2^{30}) - 1$

Returns On Success: cube number and new or current encoder coordinate, depending on whether one was provided.
On Failure: cube number and an error message

Remarks This command is used to load a new encoder coordinate for any cube. This new value is permanent only for the reference cube, however. For all other cubes, this value will be overwritten when the coordinate transformation module recalculates a new coordinate. This will happen, for any cube except the reference, when new X, Y and Z coordinates are sent to that cube, or for all cubes except the reference when the INVC command is sent to the ZY.

Example send: AZM 0, 12345
receive: AZM 1, 0, 12345

send: AZM 0
receive: AZM 1, 0, 12345

See Also CIL, COO, CX, CY, CZ, ELV, INVC

BX

Mnemonic Base X

Function Loads the X coordinate of the laser system, or requests the current X coordinate value for the laser system

Syntax BX [value]
value: new X coordinate value $-(1.7 \times 10^{308})$ to (1.7×10^{308})

Returns On Success: the current or new value of the X coordinate of the laser system.
On Failure: This function should not fail.

Remarks The location of the laser system in 3-space is represented by the set (X, Y, Z). The coordinate transformation system is dependent upon this point when it determines how to aim the laser at a particular cube. Any changes to the base X, Y, or Z values requires that the encoder coordinates for every cube but the reference be recalculated. When all changes to these values and to the other positioning offsets have been made, then the command INVC should be issued to inform every cube of the change.

Example send: BX 123.4
receive: BX 1, 123.4

send: BX
receive: BX 1, 123.4

See Also AZ0, BY, BZ, EL0, INVC, XO1, XO2, XO3, YO1, YO2, YO3

BY

Mnemonic Base Y

Function Loads the Y coordinate of the laser system, or requests the current Y coordinate value for the laser system

Syntax BY [value]
value: new Y coordinate value $-(1.7 \times 10^{308})$ to (1.7×10^{308})

Returns On Success: current or new value of the Y coordinate of the laser system.
On Failure: This function should not fail.

Remarks The location of the laser system in 3-space is represented by the set (X, Y, Z). The coordinate transformation system is dependent upon this point when it determines how to aim the laser at a particular cube. Any changes to the base X, Y, or Z values requires that the encoder coordinates for every cube but the reference be recalculated. When all changes to these values and to the other positioning offsets have been made, then the command INVC should be issued to inform every cube of the changes.

Example send: BY 123.4
receive: BY 1, 123.4

send: BY
receive BY 1, 123.4

See Also AZ0, BX, BZ, EL0, INVC, XO1, XO2, XO3, YO1, YO2, YO3

BYE

Mnemonic goodBYE

Function Gracefully shuts down socket connection to client.

Syntax BYE

Returns On Success: No return.
On Failure:

Remarks Upon receipt of this command, the ZY closes the connection to the client computer and immediately waits for a new one.

Example send: BYE
receive: nothing.

See Also QQQ, RST

BZ

Mnemonic Base Z

Function Loads the Z coordinate of the laser system, or requests the current Z coordinate value for the laser system

Syntax BZ [value]
value: new Z coordinate value $-(1.7 \times 10^{308})$ to (1.7×10^{308})

Returns On Success: current or new value of the Z coordinate of the laser system.
On Failure: This function should not fail.

Remarks The location of the laser system in 3-space is represented by the set (X, Y, Z). The coordinate transformation system is dependent upon this point when it determines how to aim the laser at a particular cube. Any changes to the base X, Y, or Z values requires that the encoder coordinates for every cube but the reference be recalculated. When all changes to these values and to the other positioning offsets have been made, then the command INVC should be issued to inform every cube of the changes.

Example send: BZ 123.4
receive: BZ 1, 123.4

send: BZ
receive: BZ 1, 123.4

See Also AZ0, BX, BY, EL0, INVC, XO1, XO2, XO3, YO1, YO2, YO3

CIL

Mnemonic Cube ILluminate

Function Commands ZY to illuminate the specified cube.

Syntax CIL <n>[, <az , el> — <x, y, z>]
n: cube name or number
az: new azimuth value $-(2^{30})$ to $(2^{30}) - 1$
el: new elevation value $-(2^{30})$ to $(2^{30}) - 1$
x: new X coordinate $-(1.7 \times 10^{308})$ to (1.7×10^{308})
y: new Y coordinate $-(1.7 \times 10^{308})$ to (1.7×10^{308})
z: new Z coordinate $-(1.7 \times 10^{308})$ to (1.7×10^{308})

Returns On Success: cube number
On Failure: error message.

Remarks <n> must be within the range of cubes specified when INI was called. CIL will return an error if it is called before INI is called, or if the cube is out of range. For instance:
INI 20; // set aside space for 20 cubes.
CIL 25; // error. CIL will report the error and do nothing.

There are 3 ways CIL can be used:

With 1 parameter: CIL assumes the parameter is the cube number, or name. It will then use previously calculated azimuth and elevation encoder coordinates to aim the servos. If INVC was previously issued, or if the cube's X, Y, and/or Z coordinates where changed since the last move (with CX, CY, and/or CZ or COO), the Coordinate Transformation Module will recalculate this cube's azimuth and elevation encoder values before they are sent to the servo controllers. The only exception to this is the reference cube.
With 3 parameters: CIL assumes the first parameter is the cube number, and that the next two are new azimuth and elevation encoder coordinates. CIL will then use these coordinates instead of the calculated ones to aim the laser at the cube. These coordinates will be wiped out the next time the Coordinate Transformation Module recalculates this cube's coordinates.
With 4 parameters: CIL assumes the first parameter is the cube number followed by new X, Y, and Z coordinates for this cube. The Coordinate Transformation Module will then use these new coordinates to calculate a new set of encoder coordinates for this cube. CIL then uses these new encoder coordinates to aim the laser at the cube.

Example send: CIL 5 receive: CIL 1, 5
send: CIL 5, -29000, -10000 receive: CIL 1, 5, -29000, -10000
send: CIL 5, 82000, 540, -100034 receive: CIL 1, 5, 82000, 540, -100034

See Also CWT

CLC

Mnemonic cube CaLCulate

Function Commands the specified cube to calculate a phase and magnitude from the acquired waveform in the DMA buffer.

Syntax CLC <n>
n: cube name or number.

Returns On Success: cube number
On Failure: cube number and an error message

Remarks CLC will fail if the cubes have not been initialised or the cube <n> is out of range or if the data in the DMA buffer does not belong to this cube. If CLC is successful, cube <n> will have updated phase and magnitude values. To obtain the digitised data for the cube <n>, use CTR, not TRG, as TRG is a low level function and the cubes would have no way to know which cube is associated with the DMA data, causing this function to fail.

Example send: CLC 3
receive: CLC 1, 3

See Also AMP, CTR, DST, PHI

CLE

Mnemonic CLear Error

Function Clears any error condition in the specified axis.

Syntax CLE <n>
n: axis number (0:azimuth, or 1:elevation).

Returns On Success:
On Failure: Axis number and error status

Remarks If an axis error occurs, an error flag is set for that axis, and the software will ignore any command to that axis until the error is cleared with the CLE command.

Example send: CLE 1 receive: CLE 1, 1

See Also

COO

Mnemonic cube COOrdinates

Function Reads/sets the specified cube's data (index, name, 3 dimensional, coordinates (X, Y, Z), and its initial encoder coordinates (az and el)).

Syntax COO <n—name>—<n—name, X, Y, Z, az, el>—<n, name, X, Y, Z, az, el>;
n : cube number.
name : ASCII string identifier for cube.
X : floating point number, part of 3D coordinate.
Y : floating point number, part of 3D coordinate.
Z : floating point number, part of 3D coordinate.
az : integer, part of encoder coordinates.
el : integer, part of encoder coordinates.

Returns On Success: cube number, name, X, Y, Z, as, el
On Failure: cube number along with an error message

Remarks Like CIL, COO has 3 distinct functions, and behaves according to how many parameters it receives:

With 1 parameter: COO assumes that the parameter is either the name or index number of an existing cube, and returns the current information (X, Y, Z, as, el) for that cube.

With 6 parameters: COO assumes that the first parameter is the name or index number of an existing cube and that the following parameters are new X, Y, Z, as, and el values for that cube.

With 7 parameters: COO assumes that the first parameter is the index number for a new cube. It deletes any cube that may already be at that index number, and creates a new cube in its place, using the remaining parameters to supply the name, X, Y, Z, as, and el for the new cube.

NOTE: The as and el parameters have been rendered obsolete by the addition of the 3D to AZ/EL coordinate transformation module. They are there for backwards compatibility, and may be set to any value. When COO responds, the values for as and el may not match those that where sent. The only exception to this is the reference cube, for which no coordinate calculations occur. Since those encoder values can be set with AZM and ELV, the as and el parameters may be eliminated in future versions.

Example To create a new cube ZG11:

send: COO 4, ZG11, -207339.04, -78346.73, 2358.98, 0, 0
receive: COO 1, 4, ZG11, -207339.040, -78346.730, 2358.980, 0, 0

To get the values for that cube:

send: COO ZG11 receive: COO 1, 4, ZG11, -207339.040, -78346.730, 2358.980, 0, 0

To change values for that cube:

send: COO ZG11, -103945.90, -45898.7, 106.34, 0, 0
receive: COO 1, 4, ZG11, -103945.900, -45898.700, 106.340, 0, 0

See Also AZM, CX, CY, CZ, ELV

CTR

Mnemonic Cube TRigger

Function Triggers A/D conversion and DMA storage of data for the specified cube and links the cube to this data.

Syntax CTR <n>
n: cube name or number.

Returns On Success: cube number
On Failure: cube number and an error message.

Remarks CTR will trigger an A/D conversion and associate this data to the specified cube. Subsequently, this data can only be used by that cube to calculate amplitude, phase and distance. CTR will fail if the system has not been initialised for cubes with INI or if the cube <n> is out of range.

Example send: CTR ZG11
receive: CTR 1, ZG11

See Also AMP, CIL, CLC, DST, PHI

CWT

Mnemonic Cube WaiT

Function Waits for servos to settle on the specified cube

Syntax CWT <n>
n: cube name or number.

Returns On Success: cube number
On Failure: cube number and an error message.

Remarks CWT will block while the laser beam is not settled on the cube. It decides whether a laser is settled on target by continuously reading the actual position of the servo. To return, it must read *n* straight readings within a specified tolerance, and within a specified time period. If CWT times out, it may still return a success status code if WMD was last used with a parameter of '0'. WTOL is used to set the settling tolerance, WCNT is used to set *n* and WTMO is used to set the time-out value. This is a cube object based command, therefore INI should have been called sometime prior to using CWT. If not, CWT will report an error.

Example send: CWT 3
receive: CWT 1, 3

Typical sequence:

```
WCNT 0, 5
WTOL 0, 5
WTMO 0, 18
WMD 0, 0
...
CIL 3
CWT 3
CLC 3
...
```

See Also WAI, WCNT, WMD, WTMO, WTOL

CX

Mnemonic Cube X

Function Loads new X coordinate value for the specified cube, or requests the current value if a new one is not provided.

Syntax CX <n>[, value]
n: cube name or number
value: new X coordinate value $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$

Returns On Success: cube number and new/current cube X value
On Failure: cube number and error message.

Remarks sending this command with a new X value will cause the az and el coordinates for that cube to be updated next time a CIL command is sent to that cube.

Example send: CX 5, 20342.24
receive: CX 1, 5, 20342.24

See Also BX, BY, BZ, COO, CY, CZ, INVC

CY

Mnemonic Cube Y

Function Loads new Y coordinate value for the specified cube, or requests the current value if a new one is not provided.

Syntax CY <n>[, value]
n: cube name or number
value: new Y coordinate value $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$

Returns On Success: cube number and new/current cube Y value
On Failure: cube number and error message.

Remarks sending this command with a new Y value will cause the as and el coordinates for that cube to be updated next time a CIL command is sent to that cube.

Example send: CY 5, 20342.24
receive: CY 1, 5, 20342.24

See Also BX, BY, BZ, COO, CX, CZ, INVC

CYC

Mnemonic CYCles

Function Requests/sets the number of cycles that the A/D system will integrate during the course of one measurement.

Syntax CYC [c]

Returns On Success: new/current cycles value.
On Failure: error message.

Remarks The value of <c> must be 4 or greater, and cycles×samples should be 65535 or less. If either condition is violated, CYC returns an error message.

Example send: CYC 32
receive: CYC 1, 32

Typical sequence:

SFQ 64 // sets the number of samples/cycle to 64.
CYC 128 // O.K., cycles×samples = 8192.
CYC // request current cycle value.

See Also IFF, SFQ

CZ

Mnemonic Cube Z

Function Loads new Z coordinate value for the specified cube, or requests the current value if a new one is not provided.

Syntax CZ <n>[, value]
n: cube name or number
value: new Z coordinate value $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$

Returns On Success: cube number and new/current cube Z value
On Failure: cube number and error message.

Remarks sending this command with a new Z value will cause the as and el coordinates for that cube to be updated next time a CIL command is sent to that cube.

Example send: CZ 5, 20342.24
receive: CZ 1, 5, 20342.24

See Also BX, BY, BZ, COO, CX, CY, INVC

DAT

Mnemonic DATa

Function Requests a range of DMA buffer data points

Syntax DAT <start, stop>
start: starting index of DMA buffer (0-65535)
stop : ending index of DMA buffer (0-65535) && start <= stop;

Returns On Success: succession of values requested, in the form
DAT 1, start + 0, val
DAT 1, start + 1, val
...
DAT 1, start + n, val
where n = stop - start.
On Failure: error message

Remarks The values returned here are the individual DMA data points, NOT the individual cycles of laser distance data (see SEQ).

Example send: DAT 0, 3;
receive: DAT 0, -1;
DAT 1, 0;
DAT 2, 5;
DAT 3, -6;

See Also SEQ

DSP

Mnemonic DeSired Position

Function Requests desired LM628 position

Syntax DSP <n>
n: axis value (0 = azimuth, 1 = elevation)

Returns On Success: axis value and the desired encoder position.
On Failure: axis value and an error message.

Remarks The meaning of the desired encoder position depends on whether the move has been completed yet. If it has, then the value returned should be the same as the target position at the time of the last STT command. If the move is ongoing, however, the desired position is the point along the projected trajectory, calculated by the LM628, that the servo should be at when the request is made.

Example send: DSP 1
receive: DSP 1, 29000

See Also ABP, ACP, RDS

DST

Mnemonic DiSTance

Function Returns the absolute distance between the specified cube and the laser instrument.

Syntax DST <n>
n: cube name or number.

Returns On Success: cube number and the absolute distance, in millimeters. This value will be within the range $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$.
On Failure: cube number and an error code.

Remarks DST will fail if the system has not been initialized for cubes, or if cube<n> is out of range. Only a standard cube can return a meaningful distance. The other cubes have the following behavior: -The reference cube returns 0. -The benchmark cube returns the survey distance, rounded down to the nearest 100 mm.

Example send: DST 17
receive: DST 1, 17, 108453.287

See Also AMP, CLC, CTR, PHI

EL0

Mnemonic ELevation 0 (zero offset)

Function Loads a new elevation encoder offset, for use with the coordinate transformation system. If the parameter is omitted, EL0 returns the current elevation encoder offset.

Syntax EL0 [value]
value: new elevation encoder offset $-(2^{30})$ to $(2^{30}) - 1$.

Returns On Success: new/current elevation encoder offset.
On Failure:

Remarks The value entered by EL0 is an angle measure, in encoder units, that is used by the coordinate transformation module to calculate the direction (in local AZ and EL encoder counts) to any cube, given that cube's absolute 3D coordinate. This value is used primarily to compensate for the location of the encoder's home position as mounted on the instrument. Despite the range of legal values that this command will accept, only angles within ± one revolution should be used. For example, if 100 000 counts per revolution encoders are being used, the value used for EL0 should range from -100 000 to +100 000.

Example
send: EL0 27000
receive: EL0 1, 27000

Subsequently:

send: EL0
receive: EL0 1, 27000

See Also AZ0, BX, BY, BZ, XO1, XO2, XO3, YO1, YO2, YO3

ELV

Mnemonic cube ELeVation coordinate

Function ELV loads a new elevation encoder coordinate for the specified cube. If no value is specified, the current value is returned.

Syntax
ELV <c>[, enc_val]
c: cube number or name
enc_val: new encoder value for azimuth LM628 $-(2^{30})$ to $(2^{30})-1$

Returns
On Success: The cube number and new or current encoder coordinate, depending on whether one was provided.
On Failure: The cube number and an error message.

Remarks This command is used to load a new encoder coordinate for any cube. However, this coordinate may subsequently change due to any changes in laser position or offsets or cube cartesian coordinates. The only cube for which the new coordinate is permanent is the reference cube. For that cube, repeating this command or issuing a CIL or COO command to that cube with new azimuth and elevation values is the only way to change the encoder coordinates.

Example
send: ELV 0, 12345
receive: ELV 1, 0, 12345

subsequently:

send ELV 0
receive: ELV 1, 0, 12345

See Also AZM, CIL, COO, CX, CY, CZ, INVC

ERL

Mnemonic ERror Limit

Function Requests/sets the error limit for the specified axis.

Syntax ERL <n>[, lim];
n: axis number (0:asimuth, 1:elevation).
lim: new error limit (0 - 25000).

Returns On Success: axis number and the new/current error limit.
On Failure: axis number and an error message.

Remarks For the LM628 servo controller chip, the position error is defined as the difference between the actual position and the desired position. ERL sets the maximum tolerable error. If this is exceeded, the servo controller will shut of the servo motor. The cause of the error can be static, i.e. something is physically preventing the servo from moving, or dynamic, where the servo motor cannot keep up with the computed trajectory. The latter is an indication of a mis-tuned servo system or an amplifier failure. A command to move to a new destination, if possible, will re-enable the servo motor.

Example send: ERL 0, 5000
receive: ERL 1, 0, 5000

send: ERL 0
receive: ERL 1, 5000

See Also ACP, AXS, DSP, RDS, STS, STW

FHM

Mnemonic Find HoMe

Function Causes the specified axis to set its home position at the index pulse position.

Syntax FHM <n>;
n: axis number (0:asimuth, 1:elevation).

Returns On Success: axis number.
On Failure: axis number and an error message.

Remarks The servo system must be homed prior to any use that requires absolute encoder coordinates. Once homed, the servo system need not be homed again unless the LM628 controller chip is reset or in the unlikely event that it should lose count. The 'homed' status of each axis can be determined by reading the system status word.
The find home operation works by moving the servo until the index pulse is detected or until it has moved one full revolution. Upon detecting the index pulse, the position counter reading is latched into the index position register and the servo is moved back to this position. When the servo has settled on this position, the position register is cleared to zero. Encoder counts can subsequently be treated as absolute encoder coordinates.
FHM attempts to complete the find home operation within 30 seconds. It can fail to do so if the servos do not move, if the index pulse is not detected within one full revolution, or if the servos cannot settle on the index pulse position. If FHM fails, or if FHM is never executed, operations that require absolute coordinates will also fail.

Example send: FHM 1
receive: FHM 1, 1

See Also STS, STW, VHM

FIL

Mnemonic Filter Integration Limit

Function Requests/sets value of the IL (Integration Limit) filter parameter for the specified axis.

Syntax FIL <n>[, f];
n: axis number (0:azimuth, 1:elevation)
f: new integration limit 0 to 32767

Returns On Success: axis number and current/new IL value.
On Failure: axis number and an error message.

Remarks

Example send: FIL 1, 50
receive: FIL 1, 1, 50

send: FIL 1
receive: FIL 1, 1, 50

See Also FKD, FKI, FKP, FLT, FSI

FKD

Mnemonic Filter KD (KD = derivative constant)

Function Requests/sets value of KD (derivative) filter parameter for the specified axis.

Syntax FKD <n>[, f];
n: axis number (0:asimuth, 1:elevation).
f: new derivative value 0 to 32767

Returns On Success: axis number and current/new KD value.
On Failure: axis number and an error message.

Remarks

Example send: FKD 0, 3000
receive: FKD 1, 0, 3000

send: FKD 0
receive: FKD 1, 0, 3000

See Also FIL, FKI, FKP, FLT, FSI

FKI

Mnemonic Filter KI (KI = integration constant)

Function Requests/sets value of KI (integration constant) filter parameter

Syntax FKI <n>[, f];
n: axis number (0:asimuth, 1:elevation).
f: new integral value (0 - 32767)

Returns On Success: axis number and current/new KI value.
On Failure: axis number and an error message.

Remarks

Example send: FKI 1, 500
receive: FKI 1, 1, 500

send: FKI 1
receive: FKI 1, 1, 500

See Also FIL, FKD, FKP, FLT, FSI

FKP

Mnemonic Filter KP (KP = proportional constant)

Function Requests/sets value of KP (proportional) filter parameter

Syntax FKP <n>[, f];
n: axis number (0:asimuth, 1:elevation).
f: new proportional value (0 - 32767)

Returns On Success: axis number and current/new KP value.
On Failure: axis number and an error message.

Remarks

Example send: FKP 0, 50
receive: FKP 1, 0, 50

send: FKP 0
receive: FKP 1, 0, 500

See Also FIL, FKD, FKI, FLT, FSI

FLT

Mnemonic FiLTers

Function Requests/sets all filter values (proportional, integral, derivative, integration limit, and derivative sampling interval) with one command.

Syntax FLT <n>[<, kp, ki, kd, il, dsi>];
n : axis number (0:asimuth, 1:elevation)
kp : proportional (0 - 32767)
ki : integral (0 - 32767)
kd : derivative (0 - 32767)
il : integration limit (0 - 32767)
dsi: derivative sampling interval (0 - 255)

Returns On Success: axis number and all the filter paramters, in the same order as above.
On Failure: axis number and an error message.

Remarks

Example send: FLT 1, 200, 300, 3500, 20, 1
receive FLT 1, 1, 200, 300, 3500, 20, 1

send: FLT 1
receive: FLT 1, 1, 200, 300, 3500, 20, 1

See Also FIL, FKD, FKI, FKP, FSI

FSI

Mnemonic Filter derivative Sampling Interval

Function Requests/sets value of SI (derivative sampling interval) filter parameter

Syntax FSI <n>[, f];
n: axis number (0:asimuth, 1:elevation).
f: new derivative sampling interval value (0 - 255)

Returns On Success: axis number and current/new SI value.
On Failure: axis number and an error message.

Remarks

Example send: FSI 1, 2
receive: FSI 1, 1, 2

send: FSI 1
receive: FSI 1, 1, 2

See Also FIL, FKD, FKI, FKP, FLT

GTI

Mnemonic Get TIme

Function Request the current value of ZY's time of day and date clock.

Syntax GTI

Returns On Success: Time in seconds since Jan 1 1970, and a time string in the form: day of week, month, day, time(hour:min:sec) and year.
On Failure:

Remarks The environment variable TZ should be set as follows:
TZ=zzz[+/-]d[d][lll]
zzz is a 3 character string represinting the name of the current time zone, and all three characters are required. example: PST (pacific std time) or EST (eastern std time). [+/-]d[d] is a required field containing an optionally signed number of 1 or more digits. This number is the local time zone's difference from GMT. Negative numbers adjust eastward from GMT. lll is an optional 3 character string (all 3 characters must be present or none at all) that represents the local timesone's daylight saving time. For instance, EDT for EST. If the environment variable TZ is missing or is not in the preceding form, then TZ=EST5EDT is assumed.

Example send: GTI
receive: GTI 1, 725665107, Tue Dec 29 16:38:27 1992;

See Also STI

IDX

Mnemonic InDeX position

Function Requests the current index position for the specified axis.

Syntax IDX <n>
n: axis number (0 or 1)

Returns On Success: axis number and index position.
On Failure: axis number and error message.

Remarks 'index position' refers to the value held in the LM628's index register. This value was latched in when the LM628 encountered an index pulse after an LM628 SINDEX command.

Example send: IDX 0
receive: IDX 1, 0, 12345

See Also FHM, VHM

IFF

Mnemonic Indtermediate Frequency

Function sets a new Intermediate Frequency for the oscillator system. If parameter is omitted, requests current IF value.

Syntax IFF [value]
value: new value for IF (.5 - 25 KHz)

Returns On Success: new IF
On Failure: error message describing error condition

Remarks This command and SFQ interact, therefore some thought must be used when using either command; IFF or SFQ can fail if (IFF value * SFQ value) exceed 100000, which is the maximum sampling rate of the Quatech DAQ-16 board currently in use. For best results, the IFF should not exceed SFQ / 4. In case of failure, these commands will leave the old values intact, so that the system will still work.

Example send: IFF 20000
receive: IFF 1, 20000

NOTE: in this case, SFQ n will work only if n <= 5. if SFQ n was issued prior to this command with an n > 5, this command will fail.

See Also SFQ

INI

Mnemonic INItialize cubes

Function Allocates enough memory for the specified number of cube objects.

Syntax INI <n>;
n: number of cube objects desired.

Returns On Success: number of cube objects allocated (allways the same number as requested).
On Failure: error message.

Remarks If this command fails for any reason other than a missing parameters, it is because it there is not enough free memory in the program heap to allocate the number of cubes requested.

Example send: INI 1600
receive: INI 1, 1600

See Also AZM, ELV, COO, CX, CY, CZ

INITZY

Mnemonic INITialise ZY

Function Initialises ZY to a known state

Syntax INITZY [d]
d: debugging flag, not useful from a remote connection as output goes to a local display. 1 turns on debug output.

Returns On Success: 1
On Failure: 0 and an error message

Remarks INITZY initialises the ZY by downloading (via FTP) from a server 2 initialisation files containing many of the commands outlined in this document. The first file, CUBES.INI, contains device independent commands, all dealing with cubes and their coordinates. The second file, ZY.INI, is distinct to each ZY and contains such device dependent information as the location and orientation of each ZY, the servo filter parameters for that ZY etc. Once the ZY has downloaded both files, it opens them (CUBES.INI first) and executes them one line at a time, much as MS-DOS executes the lines in CONFIG.SYS and AUTOEXEC.BAT on boot-up. If there where previous copies of CUBES.INI and ZY.INI on the ZY host, these are backed up. If the FTP operations fail, INITZY will return a failure code and error message, and the backup files are restored and used, so that the ZY can be initialised even if the FTP server is down. It is up to the client program to determine whether this is acceptable. If the FTP operations succeed, then the backup files are deleted.

INITZY allows all ZY's to get their initialisation from one single, easily managed source.

Example send: INITZY
receive: INITZY 1

INVC

Mnemonic INValidate Coordinates

Function invalidates the az and el coordinates for every cube in the system.

Syntax INVC

Returns On Success: Nothing
On Failure: Nothing

Remarks This command is used to force recalculation of new AZ and EL encoder coordinates next time those coordinates are needed. This recalculation may be needed as a result of changing base coordinates (location of laser system).

Example send: INVC
receive: INVC 1

See Also AZ0, BX, BY, BZ, EL0, XO1, XO2, XO3, YO1, YO2, YO3

LIMIT

Mnemonic LIMIT move

Function Establish 'software stops' on the specified axis, to limit the movement range of that axis.

Syntax LIMIT <n>[<min, max>]
n: axis number. min, max: encoder coordinates past which the servo must not go. see remarks for range of values.

Remarks LIMIT allows the movement of an axis to be confined within the 2 values, min and max. Legal values for min and max range from $-(2^{30})$ to $(2^{30})-1$, but it makes sense only to use LIMIT within ± 1 revolution of home. LIMIT is useful if there is a physical or safety reason for preventing the servo from moving within the excluded range.

Example send: LIMIT 0, -60000, 25000 //limits servo to range -60000 to 25000
receive: LIMIT 1, 0, -60000, 25000

send: LIMIT 0 // requests current limits
receive: LIMIT 1, 0, -60000, 25000

MAG

Mnemonic MAGnitude

Function Returns the last magnitude calculated by the phase calculating system.

Syntax MAG

Returns On Success: The last calculated magnitude in volts. The value will always be between 0 and 10
On Failure:

Remarks This command is distinguished from it's cube counterpart, AMP, in that it returns the latest magnitude value the phase system computed. AMP returns the latest magnitude computed for a specific cube.

Example send: MAG
receive: MAG 2.384

See Also AMP, CLC, CTR, MPC, PHI, RAD, TRG

MPC

Mnemonic Magnitude and Phase Calculate

Function Commands phase sub-system to calculate phase and magnitude from data in the DMA buffer.

Syntax MPC;

Returns On Success:
On Failure:

Remarks This command, like it's cube counterpart, CLC, commands the phase sub-system to calculate new phase and amplitude values from the acquired data. Unlike CLC, MPC will work with any acquired data.

Example send: TRG
MPC
RAD
MAG
receive: TRG 1;
MPC 1;
MAG 1, 0.351
RAD 1, 3,14159

See Also AMP, CLC, CTR, MAG, PHI, RAD, TRG

NUM

Mnemonic NUMber of cubes to scan

Function Requests/sets the number of cubes to scan.

Syntax NUM [n]
n: number of cubes to use in a scan.

Returns On Success: new/current number of cubes to scan.
On Failure: error message.

Remarks The number of cubes to scan must not exceed the maximum number of cubes as set by INI. If INI has not been called, this command will fail.

Example send: NUM 20
receive: NUM 1, 20

send: NUM
receive: NUM 1, 20

See Also INI, ORD, SCN

ORD

Mnemonic scan ORDer

Function Requests/sets the scan list, which determines the order in which the cubes are scanned.

Syntax ORD [[index][, c(1)[, c(2),.., c(n-1), c(n)]]];
index: starting point of list, must be less than total number of cubes. c(n): value of cube in n^{th} position in scan list.

Returns On Success: scan list
On Failure: error message

Remarks ORD assumes that the first parameter is an index into the scan list. All subsequent parameters are cube names or numbers that will enter the list starting at the index position. This allows a list to be initialized in many steps. For instance, if a list of 20 cubes is desired, the ORD command can be used to initialize the list in 4 steps, if 5 cubes are loaded into the list each time: The first time, index = 0, the second time, index = 4, etc. If no parameters are specified, ORD assumes an index of 0, and returns the entire list. If one parameter is supplied, ORD treats it as the index and returns the list starting from that index. The list maintained by ORD is the size specified by the NUM command.
If any of the cubes are out of range (that is, they do not exist), ORD will report failure. Although cube names can be used in place of the cube number, ORD will return the numbers of the named cubes. NOTE: A cube may be added twice to the list. However, the total number of entries in the list cannot exceed the total number of cubes.

Example The folluwing example assumes a NUM 10 command was issued:

send: ORD 0, 0, 1, 8, 5, 3 // scan cube 0 first, then cube 1, then cube 8, etc.
receive: ORD 1, 0, 0, 1, 8, 5, 3

send: ORD 5, 9, 2, 4, 6, 7 // add cubes 9, 2, 4, 6, 7 to list, starting at index = 5 receive: ORD 1, 5, 9, 2, 4, 6, 7

send: ORD
receive: ORD 1, 0, 0, 1, 8, 5, 3, 9, 2, 4, 6, 7

send: ORD 7
receive: ORD 1, 7, 4, 6, 7

Names are also possible:

send: ORD 0, ZRG, ZBG
receive: ORD 1, 0, 0, 1 // ORD does not return names, but the number of the
// named cubes

See Also INI, NUM, SCN

PHI

Mnemonic cube phase ϕ

Function Requests the last calculated phase for the specified cube.

Syntax PHI <n>
n: number of cube.

Returns On Success: cube number and the phase in radians.
On Failure: cube number and an error message.

Remarks The phase value returned differs for different types of cubes:

-Reference cube: raw phase.
-Benchmark cube: reference cube phase - raw phase
-Std cube : reference cube phase - raw phase

PHI differs from the related command RAD inasmuch as RAD returns the last phase calculated by the system, regardles of which cube it was calculated for (if any).
PHI is a cube object based command, therefore it will fail if INI has not been called or if the cube number is out of range.

Example send: PHI 2
receive: PHI 1, 2, 3.14159;

See Also AMP, CLC, DST, RAD

QQQ

Mnemonic Quit program. This command is not supposed to be easy to remember.

Function Ends ZY program, returns to DOS prompt

Syntax QQQ

Returns On Success: Nothing
On Failure: Nothing

Remarks It is not recommended that this command be used unless the program is in console mode (i.e. not driven accross the network), because there is no way to re-launch it remotely. If the program needs to be remotely restarted use RST instead (see RST).

Example send: QQQ
receive: nothing.

See Also BYE, RDF, RST

RAD

Function Returns the phase that was last calculated by the phase sub-system.

Syntax RAD

Returns On Success: last calculated phase, in radians
On Failure:

Remarks RAD is similar to PHI, but returns the last calculated phase, whether it was calculated for a cube by CLC, or calculated by MPC, a low level command. PHI returns the last calculated phase *for a specified cube.*

Example send: RAD
receive: RAD 1, 2.45833

See Also AMP, CLC, DST, MAG, MPC, PHI

RDF

Mnemonic ReaD File

Function Read a disk file located on ZY computer

Syntax RDF <filename>

Returns On Success: file, line by line, each line prefixed by 'RDF >>'
On Failure: error message

Remarks	This command is used primarily for trouble-shooting purposes. The contents of the system files CONFIG.SYS, AUTOEXEC.BAT and PCTCP.INI can be read over the network, typically in a telnet session. ZY also maintains a log of resets and their causes called REMOTE.LOG that can be read in this manner.
Example	send: RDF config.sys receive: RDF >>FILES=10 receive: RDF >>BUFFERS=10 etc.
See Also	RST

RDS

Mnemonic	ReaD Sum
Function	Reads the LM628 servo controller's integration sum.
Syntax	RDS <n> n: axis number (0:asimuth, 1:elevation)
Returns	On Success: axis number and the axis integration sum 0 to 65535. On Failure: axis number and an error message.
Remarks	The integration sum is useful for trouble-shooting positioning problems: If the sum is equal to the integration limit set with FIL, then the servo is having difficulty maintaining it's position.
Example	send: RDS 1 receive: RDS 1, 1, 50
See Also	FIL, FKI

RST

Mnemonic	ReSeT
Function	Reset ZY computer (warm boot)
Syntax	RST
Returns	On Success: Nothing On Failure: Nothing
Remarks	Causes the host (ZY) computer to perform a warm boot. Gracefully disconnects any TCP connection first.
Example	send: RST; receive: Nothing.
See Also	BYE, QQQ

SCN

Mnemonic SCaN cubes

Function Causes the cubes listed in the scan list to be measured in turn.

Syntax SCN

Returns On Success: During the scan, for each cube, SCN returns the cube number, the magnitude (in volts), the phase (in radians) and the distance (in millimeters).
On Failure: Error message.

Remarks The following commands must have been executed before a scan can take place:
INI - create the cubes;
COO - load coordinate values in cubes;
NUM - specify number of cubes to scan;
ORD - create list of cubes to scan;
FHM - homes the servos, so that encoder coordinates are no longer relative.

When scanning a list of cubes, it is advisable to include the reference cube and the benchmark cube as cubes 0 and 1 respectively in the list. This will allow the system to correct for drift in both the electronics and the index of refraction of the air. Future releases will incorporate alternate methods of determining the index of refraction of the air, but the benchmark method will always work.

Example This example assumes a scan list of {0, 1, 3}:

send: SCN receive: SCN 1, 0, 1.765, 3.14159, 0
receive: SCN 1, 1, .354, 1.53948, 98500.000
receive: SCN 1, 3, 2.085, 0.54673, 76845.498

See Also COO, FHM, INI, NUM, ORD

SEQ

Mnemonic SEQuence

Function Gets a sequence of laser ranging cycles.

Syntax SEQ

Returns On Success: the string
SEQ 1, n, amp1, ph1, amp2, ph2
for each cycle measured, where n is the cycle number, amp1 and ph1 are the amplitude and phase for A/D system 1, and amp2 and ph2 are the amplitude and phase for A/D system 2 (if installed).
On Failure: error message.

Remarks When a typical measurement is made, data is acquired over several intermediate frequency cycles. For instance, if IFF was used to set the IF to 1 KHz and CYC was used to sample 128 cycles, the measurement would integrate over 128 ms. Those cycles are then summed, and the phase and amplitude calculated from that sum. For experimental purposes, however, it may be desirable to know the phase and amplitude of each individual cycle in that string of cycles. This is what SEQ does.
The ZY instrument is capable of using 2 A/D converters, one to measure the phase of the reference oscillator, and the other to make the actual measurement. If this feature is disabled, the data values that correspond to the second A/D converter are set to 0.

Example send: SEQ;
receive: SEQ 1, 0, 3.534, 4.23489882, 0, 0 // only 1 A/D system installed
SEQ 1, 1, 3.499, 4.23398948, 0, 0
...
SEQ 1, n, 3.529, 4.23459893, 0, 0

See Also CYC, IFF, MPC, SFQ

SFQ

Mnemonic Sampling FreQuency

Function Requests/Sets the data acquisition samples per cycle

Syntax SFQ [sf];
sf: sample frequency in samples per cycle (4 - 100).

Returns On Success: current/new sample frequency.
On Failure: error message.

Remarks The meaning of SFQ has changed over time. Originally, SFQ set the frequency at which the A/D converter sampled the input signal. SFQ now sets the number of samples per cycle, with the A/D's sampling frequency being determined by $IF \times S$, where IF is the intermediate frequency, and S is the samples per cycle. This product cannot exceed 100 KHz, the maximum sampling frequency of the A/D system. SFQ will also fail if the product $S \times (cyclesmeasured)$ exceeds 65536, which is the maximum number of words the DMA system of an AT class computer can handle at one time.

Example send: SFQ 8
receive: SFQ 1, 8

See Also CYC, IFF

STI

Mnemonic Set TIme

Function sets the local ZY time.

Syntax STI <s>;
s: present time in number of seconds since 0 Hour 1/1/70.

Returns On Success:
On Failure: Error message

Remarks For this command to work, the TZ environment variable must be properly set. See GTI for an explanation of the TZ environment variable.

Example send: STI 725669478; // sets time to Tue Dec 29 17:51:18 1992
receive: STT 1

See Also GTI

STS

Mnemonic STatuS

Function Requests ZY status information.

Syntax STS

Returns On Success: following information, in order:
-The compilation date of the program (ASCII, Month Day Year)
-The compilation time of the program (ASCII, HH:MM:SS)
-The startup time of the ZY program, (seconds since 0h 1/1/70)
-The amount of free memory on the ZY computer
-A status word in hexadecimal format (see remarks)
On Failure:

Remarks The bits in the status word have the following significance:
Bit 0: (NOT)IF Phase Lock
Bit 1: (NOT)100 MHz Phase Lock
Bit 2: cube objects initialised
Bit 3: axis 0 homed
Bit 4: axis 1 homed
Bit 5: axis 0 home failed
Bit 6: axis 1 home failed
Bit 7: axis 0 verify home failed
Bit 8: axis 1 verify home failed
Bit 9: axis 0 error
Bit 10: axis 1 error
Bit 11: axis 0 motor on
Bit 12: axis 1 motor on
Bit 13: spare, always 0
Bit 14: spare, always 0
Bit 15: spare, always 0

Use STW to retrieve just the status word.

Example send: STS;
receive: STS 1, Dec 29 1992, 16:18:34, 725666027, 114432, 0x181C;

See Also STW

STT

Mnemonic STarT

Function Commands the specified axis to start a move

Syntax STT <n>
n: axis number (0:asimuth, 1:elevation)

Returns On Success: axis number
On Failure: axis number and a error message

Remarks STT will command the axis to begin moving to a target destination specified earlier by ABP. STT will also cause a new acceleration and/or velocity (loaded by ABA and ABV respectively) to take effect. This command is a low level command, used primarily for trouble-shooting purposes. CIL, the command that aims the laser at a particular cube, issues it's own STT to the two servo controllers to achieve it's purpose.

Example send: ABP 0, 50000; // new target for axis 0
STT 0; // begin the move
receive: ABP 1, 0, 50000
STT 1, 0

seealsoABA, ABP, ABV, ACP, CIL
STW

Mnemonic STatus Word

Function Requests the ZY's status word

Syntax STW

Returns On Success: the status word, in hexadecimal format
On Failure:

Remarks The bits in the status word have the following significance:
Bit 0: (NOT)IF Phase Lock
Bit 1: (NOT)100 MHz Phase Lock
Bit 2: cube objects initialised
Bit 3: axis 0 homed
Bit 4: axis 1 homed
Bit 5: axis 0 home failed
Bit 6: axis 1 home failed
Bit 7: axis 0 verify home failed
Bit 8: axis 1 verify home failed
Bit 9: axis 0 error
Bit 10: axis 1 error
Bit 11: axis 0 motor on
Bit 12: axis 1 motor on
Bit 13: spare, always 0
Bit 14: spare, always 0
Bit 15: spare, always 0

Example send: STW
receive: STW 0x181C

See Also STS

TRG

Mnemonic TRiGger

Function Triggers A/D conversion and storage into DMA buffer.

Syntax TRG

Returns On Success: Allways returns 1
On Failure: this function should not fail

Remarks TRG should not be confused with CTR, the cube object based trigger. CTR requires a parameter, the cube for which the data is being taken. This is important, because it allows the cube objects to determine the owner of the data. See below for an example. TRG does not require the system to have been initialised for cube objects to work.

Example send: TRG
receive: TRG 1

See Also AMP, CLC, CTR, DST, MAG, PHI, RAD

VER

Mnemonic VERsion number

Function Returns the ZY program's version number

Syntax VER

Returns On Success: status and the version number in the form 'Major.Minor'.
On Failure: status and an error message

Remarks None

Example send: VER
receive: VER 1, 0.3

See Also STS

VHM

Mnemonic Verify HoMe

Function Verifies home position of the specified axis.

Syntax VHM <n>;
n: axis number (0:azimuth, 1:elevation)

Returns On Success: axis number and the home position.
On Failure: axis number and an error message.

Remarks VHM works by passing the servo through the index position. If the servo was previously homed, the index pulse should occur when the position register reads 0. If not, this is a sign that the LM628 is loosing counts for some reason. Whether or not VHM reports success is not dependent on the location of the index pulse, however. VHM will report success if it can complete it's function. VHM will fail if the specified axis was not homed first.

Example send: FHM 0; // find home for axis 0.
receive: FHM 1, 0; // FHM reports success
send: VHM 0; // verifies home position of axis 0
receive: VHM 1, 0, 0; // VHM reports success, axis number
// and home position.

See Also FHM, IDX

WAI

Mnemonic WAIt for servo

Function Waits for specified servo system to settle on target.

Syntax WAI <n>
n: axis number (0 or 1)

Returns On Success: 1
On Failure: 0 and an error message

Remarks The function of WAI is to block until the servo has settled on target. How strict the definition of 'On Target' is depends on the previous use of the WCNT, WTMO, WTOL and WMD commands. In some cases where the servos *must* settle on target for an operation to succeed, WAI may fail. WAI will also fail if the axis number is out of range. For cube operations, use CWT instead of WAI.

Example send: WAI 0
receive: WAI 1, 0 // OK, servo has settled.

See Also CWT, WCNT, WMD, WTMO, WTOL

WCNT

Mnemonic Wait CouNT

Function Sets/requests the number of consecutive in-tolerance position readings used to determine if the servo is on target.

Syntax WCNT <n>[, count]
n: axis number 0 or 1
count: number of consecutive in-tolerance readings to use

Returns On Success: 1, the axis number and the new/current count
On Failure: 0 and an error message

Remarks Both WAI and CWT work by sampling the position of the servo system to see if it is at it's destination. WAI and CWT will block until a number of consecutive position readings that are within a certain tolerance of the destination is observed, or a time-out is reached. WCNT sets the number of consecutive readings, WTMO sets the time-out, WTOL sets the tolerance, and WMD determines whether WAI or CWT will report failure or success in the event of a time-out.

Example send: WCNT 1, 10
receive: WCNT 1, 1, 10

send: WCNT 1
receive: WCNT 1, 1, 10

See Also CWT, WAI, WMD, WTMO, WTOL

WTMO

Mnemonic Wait TiMe Out

Function Sets/requests the time-out value for the servo wait commands WAI and CWT

Syntax WTMO <n>[, to]
n: axis number 0 or 1
to: time out value in milliseconds, 0to(2^{30}). Large values are not recommended.

Returns On Success: 1, the axis number and the new/current time out value.
On Failure: 0, and an error message

Remarks Both WAI and CWT work by sampling the position of the servo system to see if it is at it's destination. WAI and CWT will block until a number of consecutive position readings that are within a certain tolerance of the destination is observed, or a time-out is reached. WCNT sets the number of consecutive readings, WTMO sets the time-out, WTOL sets the tolerance, and WMD determines whether WAI or CWT will report failure or success in the event of a time-out.

Example send: WTMO 1, 1000 // wait a maximum of 1000 mS for this axis to settle
receive: WTMO 1, 1, 1000

send: WTMO 1
receive: WTMO 1, 1, 1000

See Also CWT, WAI, WCNT, WMD, WTOL

WTOL

Mnemonic Wait TOLerance

Function Sets/requests the position tolerance value for the servo wait commands WAI and CWT

Syntax WTOL <n>[, tol]
n: axis number 0 or 1
tol: tolerance value in encoder units, 0 to (2^{30}). Large values are not recommended.

Remarks — Both WAI and CWT work by sampling the position of the servo system to see if it is at it's destination. WAI and CWT will block until a number of consecutive position readings that are within a certain tolerance of the destination is observed, or a time-out is reached. WCNT sets the number of consecutive readings, WTMO sets the time-out, WTOL sets the tolerance, and WMD determines whether WAI or CWT will report failure or success in the event of a time-out.

Example — send: WTOL 0, 5 // sets target tolerance to ±5
receive: WTOL 1, 0, 5

send: WTOL 0
receive: WTOL 1, 0, 5

See Also — CWT, WAI, WCNT, WMD, WTMO

WMD

Mnemonic — Wait MoDe

Function — Requests/Sets the servo wait mode for the specified axis.

Syntax — WMD <n>[, m];
n: axis number (0:azimuth, 1:elevation)
m: mode (0:loose, 1:tight)

Returns — On Success: axis number and wait mode.
On Failure: axis number and an error message.

Remarks — This determines the way both CWT and WAI operate:
-Loose: servos settle or timer times out, whichever comes first. WAI or CWT return succes in any event.
-Tight: servos must settle. Time out causes WAI or CWT to fail.

Example — send: WMD 0, 0
receive: WMD 1, 0, 0

send: WMD 0
receive: WMD 1, 0, 0

See Also — CWT, WAI, WCNT, WTMO, WTOL

XO1

Mnemonic — None

Function — Sets or requests first order laser X position constant

Syntax — XO1 [value]
where value is some experimentaly derived real number

Returns On Success: 1 and the value of XO1
On Failure:

Remarks The laser system uses the functions AZ0, BX, BY, EL0, BZ, XO1, XO2, XO3, YO1, YO2, YO3 to set values that will enable it to correctly point to a cube given only the cube's XYZ coordinates.

Example send: XO1 1.234
receive: XO1 1, 1.234

See Also AZ0, BX, BY, BZ, EL0, INVC, XO2, XO3, YO1, YO2, YO3

XO2

Mnemonic None

Function Sets or requests second order laser X position constant

Syntax XO2 [value]
where value is some experimentally derived real number

Returns On Success: 1 and the value of XO2
On Failure:

Remarks The laser system uses the functions AZ0, BX, BY, EL0, BZ, XO1, XO2, XO3, YO1, YO2, YO3 to set values that will enable it to correctly point to a cube given only the cube's XYZ coordinates.

Example send: XO2 1.234
receive: XO2 1, 1.234

See Also AZ0, BX, BY, BZ, EL0, INVC, XO1, XO3, YO1, YO2, YO3

XO3

Mnemonic None

Function Sets or requests third order laser X position constant

Syntax XO3 [value]
where value is some experimentally derived real number

Returns On Success: 1 and the value of XO3
On Failure:

Remarks The laser system uses the functions AZ0, BX, BY, EL0, BZ, XO1, XO2, XO3, YO1, YO2, YO3 to set values that will enable it to correctly point to a cube given only the cube's XYZ coordinates (see).

Example send: XO3 1.234
receive: XO3 1, 1.234

See Also AZ0, BX, BY, BZ, EL0, INVC, XO1, XO2, YO1, YO2, YO3

YO1

Mnemonic	None
Function	Sets or requests first order laser Y position constant
Syntax	YO1 [value] where value is some experimentally derived real number
Returns	On Success: 1 and the value of YO1 On Failure:
Remarks	The laser system uses the functions AZ0, BX, BY, EL0, BZ, XO1, XO2, XO3, YO1, YO2, YO3 to set values that will enable it to correctly point to a cube given only the cube's XYZ coordinates (see).
Example	send: YO1 1.234 receive: YO1 1, 1.234
See Also	AZ0, BX, BY, BZ, EL0, INVC, XO1, XO2, XO3, YO2, YO3

YO2

Mnemonic	None
Function	Sets or requests second order laser Y position constant
Syntax	YO2 [value] where value is some experimentally derived real number
Returns	On Success: 1 and the value of YO2 On Failure:
Remarks	The laser system uses the functions AZ0, BX, BY, EL0, BZ, XO1, XO2, XO3, YO1, YO2, YO3 to set values that will enable it to correctly point to a cube given only the cube's XYZ coordinates (see).
Example	send: YO2 1.234 receive: YO2 1, 1.234
See Also	AZ0, BX, BY, BZ, EL0, INVC, XO1, XO2, XO3, YO1, YO3

YO3

Function	Sets or requests third order laser Y position constant
Syntax	YO3 [value] where value is some experimentally derived real number

Returns	**On Success: 1 and the value of YO3** **On Failure:**
Remarks	**The laser system uses the functions AZ0, BX, BY, EL0, BZ, XO1, XO2, XO3, YO1, YO2, YO3 to set values that will enable it to correctly point to a cube given only the cube's XYZ coordinates (see).**
Example	**send: YO3 1.234** **receive: YO3 1, 1.234**
See Also	**AZ0, BX, BY, BZ, EL0, INVC, XO1, XO2, XO3, YO1, YO2**