LOGOSOL

Motion Control Language

MCL

Programming Guide

Language Reference

Document No 715009001 / Rev. 2.3, May 1999

MCL Programming Guide and Language Reference

Document No 715009001 / Rev. 2.3 - May1999

Logosol, Inc.

1155 Tasman Drive

Sunnyvale, CA 94089

Tel: (408) 744-0974

Fax: (408) 744-0977

http://www.logosolinc.com

Table of Contents

I. Introduction

This manual describes the features of the Motion Control Language (MCL) – a programming language used in Logosol Motion Controllers.

The manual covers the following topics:

- Control system architecture
- The host/target interface
- Controller configuration and the operating environment available to the developer
- Programming guide
- A complete MCL language reference.

An introductory chapter covering controller setup and basic command is included as well.

II. Getting Started

This chapter describes how to start the controller and establish communication with it. Examples of some basic commands, e.g. for turning the motor power ON and OFF and for starting simple movements are given.

CONNECTING THE CONTROLLER

The controller requires the following connections:

- Power supply
- Power/Signal Cable to the controlled equipment
- Emergency Power-Off Button. The button should be connected in place of the jumper shipped with the controller.
- RS-232 communication cable to the host computer. The cable should be NULL-MODEM type. This cable is connected to the male DB25 or DB9 connector at the back panel of the controller.

> ***Note***
>
> *For testing and training purposes you can connect only the RS-232 serial line and communicate with the controller without connecting it to the controlled equipment.*

Host Terminal Settings

You could use any terminal emulator program to communicate to the controller. It should have customizable settings for the communication parameters. Use baud rate set to 9600 bps, 8 data bits, one stop, and no parity control.

CONTROLLER INITIALIZATION

After the system is connected, turn on the power. Wait about 40 seconds for the controller firmware to boot-up. Press ENTER key from the terminal. You should get a prompt (>) as a response.

At the commands prompt enter the command **POWER**. It will initialize the controller boards and will turn on the power of the servo boards installed. The servo will not be activated. The digital outputs will be set to their power-on default state. Example:

```
>POWER
OK
>
```

If everything is properly initialized you'll get "OK" response. If something is wrong you'll receive an error message.

The next step of controller initialization is turning the servo control on. This is done with the command **SERVO**. Here is an example:

```
>SERVO
OK
>
```

If the response is OK the servo system is up and running. If you apply torque to the shaft of one of the controlled motors, you should fill resistance of the servo system trying to compensate this torque.

Starting a Motion

The next step is starting a motion. Assuming that the default parameters of the servo system are properly set, you need to set only the target position of the axis you want to move. This is done by setting the axis variable **ABSOLUTE**. In order to start the motion you need to issue the command **GO**. Here is an example for axis X:

```
>ABSOLUTE X = 1000
OK
>GO X
OK
>
```

You could also request a relative motion by using the command **RELATIVE**. Example:

```
>RELATIVE X = 200
OK
>GO X
OK
>
```

III. System Architecture

This chapter covers the software structure of the motion control system used in Logosol motion controllers. MCL stands on the upper level of the hierarchy of that system, but parts of it are tightly connected with lower levels. This was the motif to include a system architecture overview in this manual.

OVERVIEW

The controller firmware is based on a multitasking micro-kernel running on the top of the DOS operating system. The firmware configuration is described in an initialization file (MCL.INI), customizable for every specific application. When additional functionality is required, customized or new procedures may be built with the help of a MCL and loaded using the so-called macro file (MCL.MAC). The macro file is a piece of code that is created using MCL. Thus, user-defined programs, error handling strategies or third-party protocol emulation may be developed.

The next figure represents the top-level structure of a motion control system, which uses Logosol motion controllers and MCL:

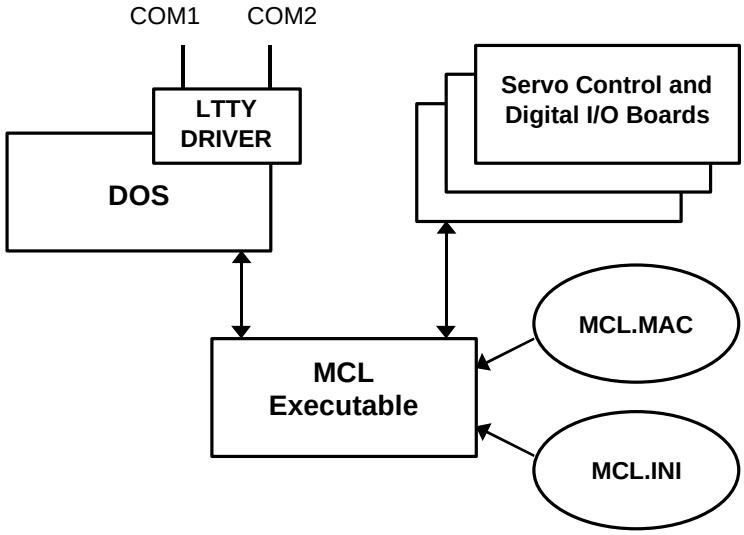

Figure 1 Firmware Structure Overview

DOS is used to provide file I/O functionality and boot-up of the system. Once started, the firmware kernel takes over the DOS and provides the real-time operation, task synchronization and axis motion synchronization.

The XTTY communication driver handles the RS-232 communication to the host computer as well as the daisy chaining multiple controller boxes to one serial interface.

System configuration is described in the MCL.INI

The main modules building the MCL executable are the dedicated real-time scheduler, command shell, motion profiler, device driver and macro processor. The diagram below describes the different modules and the connections between them:

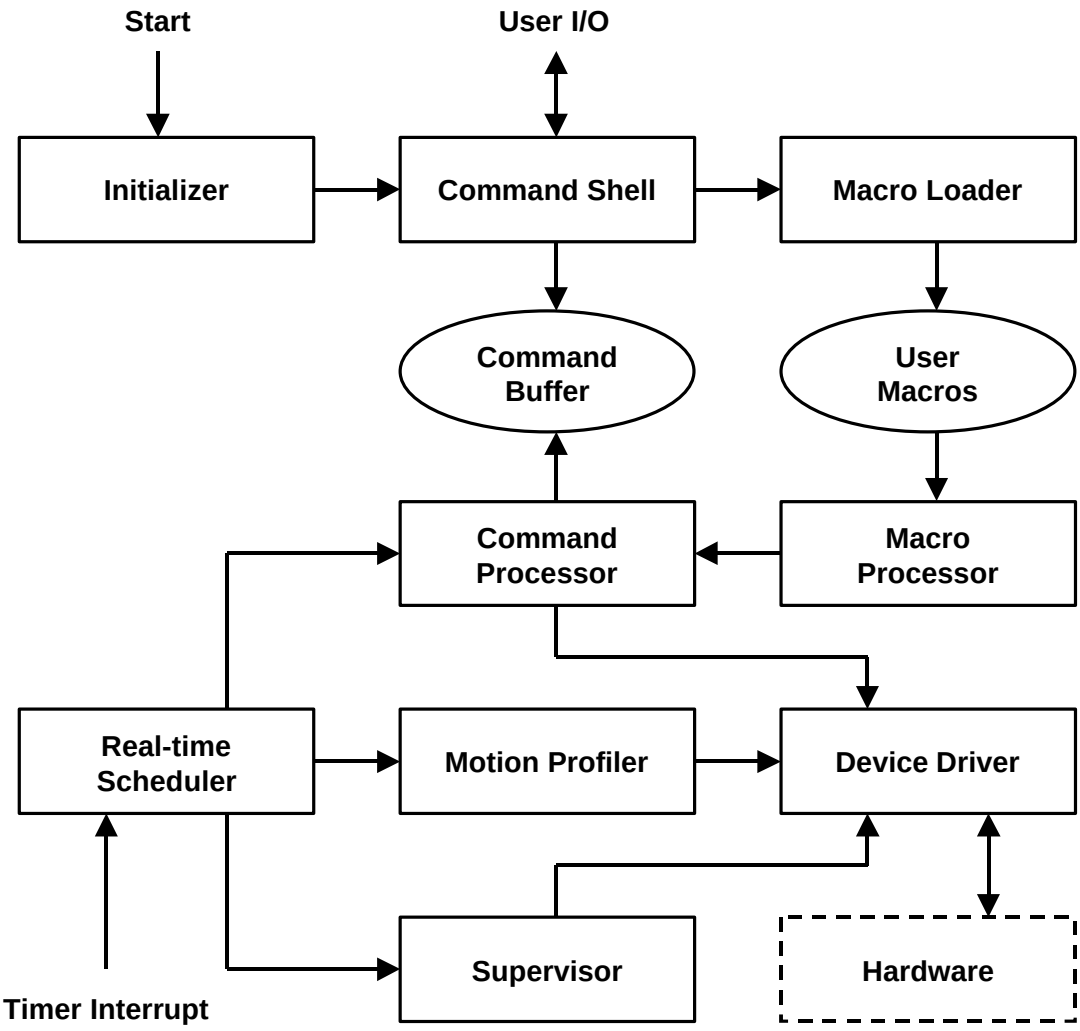

Figure 2 MCL main modules

System Initialization Procedure

The system initialization procedure consists of basic initialization, system configuration, macro file loading and executing of the startup macro.

Basic Initialization (System Boot Module)

The boot module is the first one receiving control once the firmware is started. It processes the command line parameters, allocates system resources, redirects the critical interrupts (timer, critical error, etc.) redirects the standard console to a communication driver (XTTY.SYS) and in essence takes over the DOS. Once the system is initialized the DOS is used only for file I/O operations and interface to the communication device driver.

System Configuration

The controller hardware resources have to be described in a configuration file (MCL.INI). At startup, the system is configured by processing the configuration file and creating the corresponding objects.

The configuration file describes the number and address of the installed servo control boards, the digital inputs and outputs to be used, the servo axes and their properties, the default axis parameters and the default system parameters. The initialization module creates separate object for every item declared in the configuration file. Every new object receives a name that is used as a reference to control the object.

The system settings are described at the end of the configuration file. They configure the time slice of the real time scheduler, the memory to be allocated for macros, the prompts to be used and other important parameters.

By default, the configuration file name is MCL.INI. It could be specified explicitly as a command line option. If there is an error in the configuration file the initialization module terminates the boot procedure with an error and the control is returned to DOS.

Macro File Loading

When system configuration is completed the initialization module checks for system and user macro files, specified among the command line parameters. Then, it invokes the macro loader with the given names. The processed macro files are considered separate program modules. Macro line numbers are individual for each module.

The macro loader could report insufficient memory because of the system settings in the configuration file. They define the maximum number of macros, variables and other settings.

Startup and Shutdown Macro Execution

After the macro file is loaded the system initialization module searches for a macro named “Startup”. If such a macro is found, the control is transferred to the macro processor to execute it. After the execution is complete the control is returned to the command shell that prints the first command prompt to notify the host computer that it is ready to process commands.

The Startup macro allows to have a dedicated startup sequence that initializes user defined variables, restores data, which has previously been stored in a parameter file, sets the desired supervisor, exception macros and so on.

When **QUIT** command is issued, the program searches for a macro named “Shutdown” and if it is present, the macro processor executes it before the program quits to the operating system. The purpose of this macro is to allow the host to setup an automatic cleanup before the program termination, such as halting all moving axes, determining output states, saving parameters and so on.

If no Shutdown macro have been submitted, the default behavior of the program is to issue **HALT** command without parameters, thus stopping all moving axes and terminating procedures being executed in background.

COMMAND SHELL

The command shell is the module providing the MCL user interface. It is started after the system is completely initialized and ready to process new commands. The shell indicates that it is ready to process commands by printing a command prompt.

The command shell implements the communication protocol between the host computer and the MCL kernel. Every command should consist of number of ASCII characters terminated by a Carriage Return (ASCII 13) character. After the command is processed the shell prints a prompt to indicate it is ready to process a new command. By default, the command prompt consists of three characters – ASCII 13 (CR), ASCII 10 (LF) and ‘>’. If the command is not accepted or its execution fails the command shell replaces the last prompt character with a question mark ‘?’, thus indicating to the host that the command has not been executed.

REAL-TIME SCHEDULER

The real-time scheduler is responsible for sharing the CPU time between the tasks running concurrently. It also provides the hard real-time performance of the controller meaning the operation of the tasks does not depend on the current load of the system and is totally predictable.

The real-time scheduler implements the concept of time-slices defined by the timer interrupt. By default, the time slice is set to 6 ms, but it could be changed in the system section of the configuration file. Every time slice the scheduler receives control and starts all real-time tasks one by one. They are designed to be non-preemptive so their execution normally ends before the expiration of the time slice. The remainder of the time in the time slice is available to non real-time tasks like the target/host communication and the user interface.

The real-time tasks running concurrently in the controller are Foreground, Background, System Check, Supervisor, Motion Profiler and Brakes control tasks. They are briefly described in the following subsections.

Foreground and Background Tasks

The Foreground task does the interpretation of the host commands. It also takes care for transmitting the messages between the Background and the Foreground tasks.

Both Foreground and Background tasks can process macros. The macros can be started by the command shell or from another task. Every time slice a single line from the macro is executed.

When the command shell receives a command that is a macro name, the command processor uses the Foreground task to execute it. No prompt is returned to the user until the macro is completely executed. The return value set by the return statement of the started macro determines whether the command shell will print acknowledge (CR, LF, '>') or not acknowledge (CR, LF, '?') prompt.

When the host needs to communicate with the system during a procedure that might take quite a long time to complete, the procedure should be executed in the Background task. Using the Submit statement from within a macro or from the command line instructs the Real Time Scheduler to execute the procedure in the Background task. The Background task is running even if another macro is started in the Foreground task. Thus, the execution status of a long procedure could be monitored or interrupted. Only one Background task can run at a time.

System Check Task

Every time slice, the MCL performs a series of administrative and supervising actions to verify the system integrity:

- Checks for the emergency power off signal.
- Checks the servo amplifiers for current overload.
- Checks whether axes positions are within the range defined by the software limit positions.
- Checks whether the tracking error exceeds the limit for maximum position error.

If any of the above conditions is violated a dedicated exception handler macro is started. It is executed in the background task. Thus, if the background task is running and an exception occurs the currently processed macro is interrupted and the exception handler macro is started instead.

Supervisor Task

The supervisor task is started every time slice. You could specify a macro to customize the supervisor task. This macro will be executed every time slice in its entirety. The supervisor task is designed to perform recurring tasks like controlling input states or setting output states. It could also check the consistency of the system based on user-defined conditions and eventually perform corrective actions.

Any of the macros defined could be set as Supervisor macro but not all MCL commands may be executed within the Supervisor macro. Commands that take undetermined time to execute e.g. file I/O and user interface commands are not allowed.

The Supervisor macro should be designed to consume as little time as possible during normal operation. The supervisor macro is installed with the command Supervisor.

Motion Profiler Task

The Motion Profiler Task runs the motion profile generator. It provides the synchronization of multiple motors in accordance with the desired motion path. Every time slice the motion profile generator loads new desired velocity to each of the motors participating in the movement.

The tracking error heavily depends on the time slice length. The shorter is the time slice, the smaller should be the tracking error. Other factors affecting performance are the dynamics of the controlled mechanical system, as well as the maximum speed and acceleration set for each axis.

Brakes Control Task

The Brakes Control Task is intended to control the status of brakes attached to the shaft of the controlled motors.

When a motion is started the brake attached to the corresponding motor is released. After the motion completes the brake is activated after a specified time-out. If another motion command is issued before the time-out expiration the timer is reset.

MACRO PROCESSOR

The Macro processor is responsible for the interpretation of commands within the currently executed macros. It processes the commands related to program flow, handles run-time errors, and maintains the history buffer by keeping track of the started macros and the detected exceptions.

Program Flow Control

The program flow control module maintains program execution status, the program line counter and the stack associated with every macro processing task. Normally, it increases the program line counter by one after every executed command. However, if the command is one of the statements controlling the flow control like **IF - ELSE - ENDIF** or **GOTO** then the next value of the program counter is set by the result of the execution of these statements.

Error Handling

If for any reason a program line can not be executed (for example division by zero, or use of invalid index for an array) the normal program execution is interrupted. The content of the program counter is stored in a dedicated variable and an appropriate message is stored in the history buffer. Based on the system configuration an error message might be displayed.

Exceptions

Exceptions are emergency events that might occur at any time no matter if a macro is executed or a motion is started. Exception generators are built-in system integrity checks as well as user defined conditions. Whenever an exception event is generated it is stored in the history buffer, the normal execution of the background task (if started) is interrupted and a dedicated macro is started. The user could define this exception macro in order to implement error specific actions. If there is no exception defined, the MCL kernel provides default actions to ensure system safety.

History Buffer

The history buffer is intended to keep track of the activities of the macro processor and the detected exceptions. It is implemented as a circular buffer that keeps the 20 most recent commands. The content of the buffer could be displayed through the RS-232 line or stored to a file inside the controller for later reference.

COMMANDS PROCESSOR

The command processor executes the commands submitted by the user through the command shell of the macro processor, executing statements from a started macro. It returns information about the execution status and the value of the program line counter.

If the command is invoked from the command shell and it fails, then the command shell returns “not acknowledged” prompt to the user (‘?’ by default). If the command is invoked from the macro processor and it fails, then the macro execution is terminated.

Normally all commands are executed within a single time slice. However there are exceptions like the GO command that might have to wait for the brake to release the shaft of the motor if needed.

MOTION PROFILER

Motion Profiler module generates the velocity profiles and provides the synchronization of the participating axes.

The motion profiler is started by the **GO** command and keeps running until the moving axes reach their desired position. It also terminates if one or more of the moving axes stops because of an error.

DEVICE DRIVER

The motion control device driver provides unified interface to the different motion control boards that the controller could be configured with. The user does not have direct access to the driver functions and its internal structures.

The driver is an integral part of the firmware executable. It is linked with the rest of the modules in order to avoid any communication overhead.

IV. Host Computer Interface

RS-232 Interface Cable

RS-232 interface cable has to comply with the specification of “NULL Modem” cable.

Serial Interface Parameters

The serial interface parameters are configurable by the communication driver. By default they are:

Speed: 9600 baud per second

Data bits: 8

Stop bits: 1

Parity check: none

The maximum communication speed supported is 115200 baud per second.

Communication Protocol

The communication protocol consists of ASCII strings. The commands from the host to the controller are terminated with Carriage Return [ASCII 13]. The response from the controller is terminated by Carriage Return [ASCII 13], Line Feed [ASCII 10] and Prompt character. If the command is accepted, then the Prompt character is Greater Than Sign [>]. If the command has a syntax error, or its parameter is out of range, or it can not be executed for some reason then the Prompt sign character is Question Mark [?]

Here is an example of a communication session:

Host	Controller	Comment
POSITION X [CR]		The host requests the current position of axis X
	1000[CR][LF]>	The controller returns the position and prompt
POSITION X = 100[CR]		The host sets the current position of axis X
	[CR][LF]>	The controller confirms the command is accepted
METALLICA X[CR]		The host sends wrong command
	[CR][LF]?	The controller indicates invalid command

SYNCHRONIZATION WITH HOST COMPUTER

The synchronization with the host computer is needed to ensure the commands are sent at the proper time, and in the desired sequence. This is especially important for commands executing motion and taking relatively long time to complete. The synchronization problem could be resolved in two different ways.

The first approach is synchronous – the host should poll the controller status and wait until it is busy. The next motion command should be sent after the controller reports “ready”. The advantage of this method is that the current controller status is available to the host at all times. The disadvantage of this approach is the communication overhead.

The second approach is asynchronous – the host starts the motion and waits for notification message from the controller. At the end of the started motion the controller sends the message to the host. The advantage of this synchronization method is the smaller communication overhead. A negative side to it is that the host has an update of the status of the controller only at the end of every motion.

V. Controller Configuration

Logosol Servo Controller provides flexible utilization of the system resources. The configuration information is described in a initialization file (by default, MCL.INI), used by the Motion Control Kernel during its startup. This chapter describes the format and contents of this file.

The name of the configuration file is specified as a command line parameter for MCL.EXE. This allows you to have a number of different configurations stored in the controller at the same time. This could be helpful during development and testing or if different machines need to be supported by the same controller.

During startup, the Motion Control Kernel reads the file line by line and creates internal structures of data dedicated to every described resource. If an error is detected, the startup procedure prints an error message and the line number where the error occurred, then terminates its execution.

The configuration file consists of sections defining controller resources and their parameters. The names of the sections are: BOARDS, INPUTS, OUTPUTS, AXES, PARAMETERS and SYSTEM. The syntax and the parameter settings of these sections are described next.

DEFINING BOARDS

The first section defines the boards in the controller. It deals with the servo and the digital I/O boards that the user wants to declare to the firmware (other boards inside the controller - CPU board, disk emulator board, etc. are not declared). A name is associated with each board installed at a given base I/O address. Boards, which are present in the controller box but are not declared, are neglected by the firmware.

Syntax:

```
[BOARDS]
BoardName       IOAddress
```

Parameters:

BoardName	Any identifier to refer to the board
IOAddress	The base I/O address to which the board is set up.

The name of the board must start with an alphabetic character and should not exceed 40 characters.

The base address could be specified as a decimal or hexadecimal number. The hex number should start with 0x prefix. Example: 0x280. The specified base address must correspond to the jumper settings of the board. Refer to the board documentation for details regarding the base address setup of the particular board.

Example:

```
[BOARDS]
XYSTAGE         0x280
```

DEFINING DIGITAL INPUTS

The Inputs section defines the digital inputs to be used. Each input line is identified by its name, followed by the board name to which it belongs, an input index and the active state of the input corresponding to a logical 1.

Syntax:

```
[INPUTS]
InputName       BoardName       InputNumber       ActiveState
```

Parameters:

InputName	Name assigned to the input line
BoardName	Name above defined board
InputNumber	Hardware index of the input line in the board
ActiveState	LOW or HIGH level to be considered logical 1

The name of the input must start with an alphabetic character and should not exceed 40 characters.

The *BoardName* should be one of the boards defined in the [BOARDS] section. The configuration file processor does not process forward references and that is why the [INPUTS] section has to be defined after the [BOARDS] section.

The *InputNumber* is the index of the input you are defining within the specified board. Refer to the board documentation about the correspondence of the input numbers and the pins of the interface connectors.

By defining an *ActiveState* the desired polarity of the input can be changed easily. The possible settings of this field are “LOW” or “0” and “HIGH” or “1”.

The maximum number of input names that can be defined is 80.

Examples:

```
[INPUTS]
; InputName      BoardName    InputNumber    ActiveState
HomeSwitchX      XYZ          IN19           LOW
HomeSwitchY      XYZ          IN17           LOW
HomeSwitchZ      XYZ          IN15           LOW
VacSensor1       XYZ          IN7            LOW
VacSensor2       XYZ          IN8            LOW
```

DEFINING DIGITAL OUTPUTS

The Output section defines the digital outputs. Each output line is identified by its name, followed by the board name to which it belongs, an output index, the active state of the output corresponding to a logical 1 and the “power on default” state of the output.

Syntax:

```
[OUTPUTS]
OutputName      BoardName      OutputNumber      ActiveState      PowerOn
```

Parameters:

OutputName	Name assigned to an output line
BoardName	Name assigned to the board
OutputNumber	Output line number on the board
ActiveState	LOW or HIGH to be logical 1
PowerOn	The state after POWER command

The name of the output must start with an alphabetic character and should not exceed 40 characters.

The *BoardName* should be one of the defined boards in the [BOARDS] section. The *OutputNumber* is the index of the output you are defining within the specified board. Refer to the board documentation about the correspondence between the output numbers and the pins of the interface connectors.

By defining an *ActiveState* the desired polarity of the output can be changed easily. The possible settings of this field are “LOW” or “0” and “HIGH” or “1”.

The *PowerOn* field specifies the state of the output after the execution of POWER command.

> ***Note***
>
> *The outputs controlling the amplifier and sensor power must have PowerOn state HIGH in order to make possible turning the amplifier power on.*

Refer to the documentation of the specific servo control board for the outputs that must be turned on. The maximum number of output names that can be defined is 80.

Examples:

```
[OUTPUTS]
; OutputName  BoardName    OutputNumber    ActiveState  PowerOn
CloseValve1   XYStage      OUT0            HIGH         OFF
CloseValve2   XYStage      OUT1            HIGH         OFF
PowerOPS      XYStage      OUT8            HIGH         ON
PowerMain     XYStage      OUT10           HIGH         ON
BrakeZ        XYStage      OUT13           HIGH         OFF
```

DEFINING AXES

The AXES section defines the servo-controlled axes. Each axis is represented by multiple line section describing different axis parameters. The section starts with the parameter NAME and ends with the beginning of a new section (with its own NAME parameter) or with the beginning of a new configuration file section.

Syntax:

```
[AXES]
Name            AxisName AxisType
Master          BoardName AxisIndex
Slave           SlaveAxis LinkCoeff
Encoder         CountsPerRev
Transmission    TransmissionRatio
Brake           OutputName Time-Out
```

The following parameters are specific for brushless motors only:

```
HallSense       LogicLevel
InitPower       InitPowerLevel
InitTime        InitTime
Poles           PolesNumber
Phases          PhasesNumber
```

The table below describes the meaning of the different parameters:

Parameter	Arguments	Description
Name	AxisName AxisType	This parameter defines an axis with the name specified as the first argument. The identifier should start with the alphabetical character and should not exceed 40 characters. After the name you could specify linear [L] or rotational [R]. This argument is optional. By default all axes are linear. The type of axis is important for using different scaling coefficients. They are individual for the linear and the rotational axes. In that way you may switch between inches and meters without changing the measurement unit for the rotational axes, i.e. degrees. The *Name* parameter initiates new axis definition section.
Master	BoardName AxisIndex	This is mandatory parameter that associates the specified above axis name with particular board and index within the board. The Logosol servo boards support from 1 to 4 servo channels and they are numbered with indexes from 0 to 3.
Slave	AxisName LinkCoeff	The *Slave* parameter defines an axis as a dependent of another axis, called master axis. When the master axis moves to a given distance, the slave axis moves automatically to the same distance multiplied by the link coefficient. Note that a single master axis can have multiple slave axes. They should be specified on a different lines starting with the *Slave* parameter.
Encoder	CountsPerRev	The *Encoder* parameter specifies the resolution of the encoder installed on the motor moving the given axis. Note that the resolution is specified as “raw” resolution, i.e. the number of encoder counts per revolution for one of the encoder phases, This number is indicated also in the encoder spec. Logosol motion controllers use quadrature encoder signal reading, meaning that the raw encoder resolution is multiplied by 4.
Transmission	TransmissionRatio	The *Transmission* parameter specifies the number of turns of the transmission output shaft for one revolution of the motor shaft. This parameter along with the *Encoder* parameter is used by the firmware to calculate the correspondence between the user defined units (for example thousands of an inch) and the number of encoder counts generated for this motion. The *Transmission* coefficient could be specified with very high accuracy. The firmware will process a floating point number with up to 15 significant digits.

Parameter	Arguments	Description
Brake	OutputName Time-Out	The *Brake* parameter supports the motors that have brakes controlled by some of the controller digital outputs. The parameter defines one of the output lines as such an output. Every time the motor has to move the firmware will turn on this output effectively releasing the brake. If the motor does not move for the specified time-out period the output will turn off the brake and this will stop the motor shaft. Note that the brake output should be powered by OPS outbound power source in order to turn the output to low automatically (by the hardware) when the motor power amplifiers are disabled. Thus, whenever the motor amplifiers are overloaded and disabled by hardware, the brake power is turned off and the brake will be activated automatically (provided the brake is normally closed), with no need for software support.
HallSense	LogicLevel	The *HallSense* parameter sets the logic level interpretation for the hall sensor inputs of brushless motors. This parameter is only applicable for configuring brushless motor servo controllers.
InitPower	InitPowerLevel	This parameter sets the initial value for motor command register, used when executing algorithmic phase initialization of brushless motors. This parameter is only applicable for configuring brushless motor servo controllers.
InitTime	InitTime	The *InitTime* parameter sets the amount of time to wait after phase initialization of brushless motors in units of servo chip time slices. This parameter is only applicable for configuring brushless motor servo controllers.
Poles	PolesNumber	The *Poles* parameter sets the number of encoder counts per commutation cycle of brushless motors. This parameter is only applicable for configuring brushless motor servo controllers.
Phases	PhasesNumber	The *Phases* parameter sets the current commutation waveform according to number of phases of the motor. There are 2 and 3-phase brushless motors. This parameter is only applicable for configuring brushless motor servo controllers.

Examples:

```
[AXES]
Name            X                ; Axis name (by default - linear)
Master          XYStage 0        ; Master board and axis index
Encoder         2500             ; Encoder counts

Name            Rotor R          ; Indicate rotational axis
Master          XYStage 2
Slave           X –1.5           ; Slave axis name and ratio
```

SETTING STARTUP PARAMETERS

The [PARAMETERS] section is intended to provide set of default or start-up parameters for the axes. These parameters are set before any command is processed. They ensure that the axes are in known, defined state after start up.

Syntax:

[PARAMETERS]
AxisName AxisParam = Value

Example:

```
[PARAMETERS]
VEL     X = 400     Y = 400     Z = 400
ACC     X = 1000    Y = 1000    Z = 1000
MAX     X = 1000    Y = 1000    Z = 1000
KP      X = 80      Y = 80      Z = 80
KI      X = 100     Y = 100     Z = 100
KD      X = 2000    Y = 2000    Z = 2000
IL      X = 40      Y = 40      Z = 40
DS      X = 256     Y = 256     Z = 256
FLIMIT  X = 5000    Y = 5000    Z = NOLIMIT
RLIMIT  X = -2000   Y = -2000   Z = NOLIMIT
```

SYSTEM SETTINGS

The section [SYSTEM] is used to define the values of the system specific parameters. They control the time-slice, the response mode, the memory distribution, the names of the external programs implementing the file transfer protocols, the command shell prompt and so on. The table below describes all parameters and their meaning.

Syntax:

[SYSTEM]
ParName *Value*

Parameter	Description
TimeSlice	The period between two subsequent cycles of the real-time kernel. The dimension is 1 millisecond. The maximum setting is 55 ms.
Info	The format of the status returned. The allowed values are: • NONE no status messages are returned unless requested • DEC status is returned as decimal number • HEX status is returned as hexadecimal number • TEXT status is returned as a set of text messages
Report	An integer number that represents a bit mask for the reporting option; see *Report* variable for a description of the mask; the value is the default for the Report variable
MaxSamples	The maximum number of samples to be recorded by the **RECORD** command. It stores the current position and error of the specified axis.
MaxMacros	The maximum number of macro definitions in one macro file (default = 100)
MaxProcs	The maximum number of procedure definitions in one macro file (default = 100)
MaxLabels	The maximum number of labels defined in a macro file (default = 50)
MaxLines	The maximum number of lines in one macro file (default = 1000)

MaxVars	The maximum number of variables defined in a macro file (default = 100)
MaxSamples	The maximum number of position samples (default = 0)
Upload	Name of an executable file (. COM or .EXE) to be called by the **UPLOAD** command to upload files to the host computer.
Download	Name of an executable file (.COM or .EXE) to be called by the **DOWNLOAD** command to download data.
PromptACK	Prompt to be returned if the user command is accepted. The definition consists of a list with the ASCII codes of the prompt. Note that (because of the DOS environment) ASCII code 10 (Line Feed) is transformed automatically to 13, 10 (Carriage Return, Line Feed).
PromptNAK	Prompt to be returned if the user command fails. The definition format is identical to the one for *PromptACK* parameter.
Kernel	ON or OFF – turns on (default) or off the kernel level command set. When the kernel commands are disabled, the syntax analyzer recognizes only the macro names as reserved words. The rest of the commands are available with dot prefix.

CONFIGURATION FILE TEMPLATE

```
[BOARDS]
XYZ             0x0280

[INPUTS]
HomeSwitchX     XYZ    IN0    LOW
HomeSwitchY     XYZ    IN1    LOW
HomeSwitchZ     XYZ    IN2    LOW

[OUTPUTS]
Pwr             XYZ    OUT0   HIGH   ON
Pwr40V          XYZ    OUT1   HIGH   ON
BrakeZ          XYZ    OUT2   HIGH   OFF

[AXES]
Name            X                          ; Axis name
Master          XYZ 1                      ; Master board and axis index
Scale           1.5                        ; Scaling coefficient
Encoder         1800                       ; Encoder counts

Name            Y
Master          XYZ 0
Slave           X -1                       ; Slave axis name and ratio
Scale           100
Encoder         1800

Name            Z
Master          XYZ 2
Scale           300
Encoder         1800
Brake           BrakeZ       30000         ; Brake output and timeout

[PARAMETERS]
VELOCITY        X = 400      Y = 400       Z = 400
ACC             X = 1000     Y = 1000      Z = 1000
MAX             X = 1000     Y = 1000      Z = 1000
KP              X = 80 Y = 80 Z = 50
KI              X = 100      Y = 100       Z = 100
KD              X = 2000     Y = 3000      Z = 2000
IL              X = 40 Y = 40 Z = 40
DS              X = 256      Y = 256       Z = 256
FLIMIT          X = NOLIMIT  Y = NOLIMIT   Z = NOLIMIT
RLIMIT          X = NOLIMIT  Y = NOLIMIT   Z = NOLIMIT

[SYSTEM]
TimeSlice       4
Info            Text
Report          0xffff
Download        transfer /r
Upload          transfer /s
```

VI. Motion Control Language Structure

A specialized programming language, called Motion Control Language (MCL), controls the servo controller resources. Using MCL, which can handle various control tasks, Logosol controllers may be implemented in a wide range of applications. Common applications include robot control and industrial automation.

SYNTAX

The MCL is an ASCII command language. Its character set is made of printable characters, CR, LF and EOF only. The next subsections cover the following topics:

Character set

Identifiers

Data Types

Labels

Unary, binary and bitwise operators

Operator precedence

Comments

Multi-axis syntax

Character Set

Letters	A – Z, a - z
Digits	0 1 2 3 4 5 6 7 8 9
Special Characters	"#%&'()*+,-./:;<=>_ | ! $? @ [\ ] ^ { }
Space	ASCII 32
Comma	ASCII 44 ,
Carriage return	ASCII 13 (^M)
Line feed	ASCII 10 (^J)
EOF	ASCII 26 (^Z)
Underscore	ASCII 95 _

Identifiers

An identifier is a sequence of alphanumeric characters. It must start with a letter or with underscore (_) character. The maximum length should not exceed 40 characters.

Examples:

Axis_1	correct
This_is_an_identifier	correct
1st_ax1s	not correct, must start with a letter

Data Types

The only available data type is a 32-bit signed integer.

If a number starts with 0x it is interpreted as a hexadecimal.

Examples:

12345	valid integer number
0x100	hexadecimal integer, equals decimal 256,
12345.00	not valid, must not have a fractional part

Labels

The label marks a certain position within a macro. An identifier followed by a colon defines it. The labels are used as arguments by **GOTO** statements.

Examples:

```
Loop1:                    ; Label definition in a separate line
ABS X=1000
GOTO Loop1                ; Jump to the label – infinite loop

Loop2: ABS X=1000         ; Label definition together with a command
; more commands ...
GOTO Loop2                ; Jump to the label
```

Unary Operators

-	Negation
!	Boolean NOT
~	Bitwise NOT (inversion)

Binary Operators

+	Add
-	Subtract
*****	Multiply
/	Divide
&&	Boolean AND
||	Boolean OR
<	Less Than
<=	Less or Equal
>=	Greater or Equal
>	Greater
!=	Not equal
==	Equal
=	Assignment

Bitwise Operators

The bitwise operators work on the individual bits of the operands. They are useful for setting or clearing specific bits within a variable.

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

Operator Precedence

All operators are processed with dominating priority as shown below:

1. * /
2. + -
3. <= =>
4. != ==
5. &
6. ^
7. |
8. &&
9. ||
10. =

Comments

Every string found in a command line after a semicolon is treated as a comment.

Examples:

```
; This is a comment
ABS X=100            ; The comment starts with semicolon
```

Multi-Axis Syntax

The multi-axis syntax allows setting an axis parameter for multiple axes in a single line statement. It also provides report of parameter settings or applying a motion command to a number of axes. This syntax has two major advantages – it simplifies the manual entry of data and provides synchronous update of the parameters (or execution of the motion commands). While the ease of use is important only if commands are entered manually, the synchronous update is important in both macro processing and

user command processing. The performance improvement comes from the fact that the motion control kernel executes one command line per time slice. If you need to update an axis parameter or start multiple axes at the same moment the single axis syntax will result in several lines of code. In this case it is appropriate to use the multi-axis format of the commands.

The common format of multi-axis parameter setting is following:

ParName AxisName1 = Value1, AxisName2 = Value2, … AxisNameN = ValueN

The above syntax is equivalent to the following set of commands issued as separate commands:

ParName AxisName1 = Value1

ParName AxisName2 = Value2

. . .

ParName AxisNameN = ValueN

The maximum command line length limits the number of axes set in a single line.

Example:

```
VEL X = 1000, Y = 1200
ACC X = 2000, Y = 3000
```

The multi-axis syntax for the commands reporting data has the following format:

ParName AxisName1, AxisName2, … AxisNameN

The single command equivalent is:

ParName AxisName1

ParName AxisName2

. . .

ParName AxisNameN

Example:

```
VEL X, Y
ACC X, Y
```

The last form of the multi-axis syntax concerns the axis control commands like GO, HALT, LATCH etc. Normally if you don't specify axis parameter to these commands the command interpreter assumes that the command is applicable to all axes. If you specify explicitly the axis name then motion command will apply to the specified axis only.

The multi-axis syntax allows combination of the above alternatives. You could specify a list of axes you want this command to be applied to. Here is the syntax format:

CmdName AxisName1, AxisName2, … AxisNameN

The single command equivalent is:

CmdName AxisName1

CmdName AxisName2

…

CmdName AxisNameN

Example:

```
GO X, Y
```

LANGUAGE ELEMENTS

Variables

A variable holds the actual value of a parameter that can be redefined during operation of the MCL. Each variable can be used as a function as well.

Some variables are actually a collection of variables, called arrays. The most important one is the axis variable. The individual items of an array can be requested by writing the identifier of the array element. If a value is assigned to an array, the assignment is actually done to the array element.

The array indexes can be surrounded by square brackets.

Examples (we assume three axes - X, Y and Z):

```
VELOCITY X=1000 Y=2000
VEL [X]=1000 [Y]=2000
VEL [X] [Y]
```

The MCL allows the declaration of user variables via the VAR statement. Once declared, user variables are used in the same way as predefined variables. Each user variable implicitly defines a virtual function with the same name. The following types of variables exist:

- Multi-axis variables (referenced with names or indexes)
- System Variables
- Read-only variables
- Task-dependant variables
- User-defined variables

Constants

The MCL allows the declaration of user constants with the CONST statement. Once declared, user constants can be used in the same way as predefined functions.

Definitions

The MCL language provides a statement DEFINE that allows you to associate an identifier with expression. If later in the program the defined identifier is used it will be replaced (at run-time) with the defined expression.

Procedures and Macros

An MCL program consists of data definitions and a set of procedures and macros. They are called by specifying their name followed by the required parameters. This could be done either from the command line or within the program (calling of subroutine).

Macros and procedures differ in the way they are processed. If the user calls a macro from the command line, the prompt will be returned after the end of execution. This means that the user has no control over the system during the macro execution.

If user calls a procedure from the command line then the procedure execution is started (submitted) to the background task and a prompt is returned to the user immediately. After that the user can issue an MCL command or even a macro, which will be executed in parallel with the procedure. The only limitation is that no second procedure can be submitted before the end of the first one.

It is possible to call procedures and macros from other procedures and macros. In this case the called subroutine is processed by the same task as the calling one, unless a procedure or a macro running in the foreground task “submits” a procedure to the background task. Thus, the type of the routine - a macro or a procedure is of significance only during the command line processing.

Flow Control Commands

The MCL program statements are used for defining procedures and macros, specifying constants and variables and managing execution flow control.

IF-ELSE-ENDIF	Specify conditional branch
WHILE-ENDWHILE	Define loop
GOTO	Unconditional jump to label
WAIT	Hold program execution until expression becomes TRUE
DELAY	Hold program execution for a specified time
SUBMIT	Start procedure execution as a concurrent task
RETURN	Indicate end of macro or procedure

Labels needed as a parameter for the **GOTO** statement are defined by semicolon ":" at the end.

VII. Programming Guide

SYSTEM INITIALIZATION

After the controller is powered up, the firmware is loaded and different modules are initialized. As a part of the initialization process the firmware loads the macros and looks for a macro called STARTUP. If such a macro exists it is executed.

The purpose of this macro is to initialize the variables, restore the calibration parameters from a file and eventually notify the host computer that the controller is up and running. Below is an example of a simple startup macro:

```
MACRO Startup
EXEC params.dat       ; Restore saved parameters
PRINT "Ready"
RETURN
```

If an embedded system is developed that has to start as soon as the power is up then this is the macro that should take care for initializing the hardware, turning the servo control on and begin execution of the working cycle. Here is an example of such initialization:

```
MACRO Startup
EXEC params.dat
POWER
SERVO
SUBMIT MainLoop       ; Start procedure in the background task
RETURN
```

AXIS POSITION LATCHING

The position latching is a process of recording the current axis position in a dedicated "index" register. The latching is synchronized by the hardware with the rising or falling edge of a selected external signal. This signal could be either the index of the encoder (when an axis is homed) or any of the available external strobe signals.

The selection of the strobe signal is implemented with the axis parameter *Strobe*. The code to be set should be selected from the table corresponding to the board you are using. Look at the description of the command Strobe in the command reference for the current tables. Note that the code also determines if the latching should be done on the falling or on the rising edge of the strobe signal.

The start (enabling) of the latching procedure is initiated with the command *Latch*. It clears the index register and sets the bit 0 (mask 0x0001) in the status word. It indicates that the latching is initiated. After the position is latched the bit 0 is cleared and bit 3 (mask 0x0008) is set to indicate the latching is completed. No other position will be recorder until a new *Latch* command is issued.

The position latching is a precise way to measure the coordinate of an axis at the moment the strobe signal is activated. That is why the latching is used in the home procedure, in scanning procedures – always when an accuracy is needed.

Here is an example of how latching is initialized and started:

```
; Procedure Name:
;        LatchPos
;
; Parameters:
;        Axis  – name if the axis to be homed
;        Scode – code of the index strobe signal
;
; Calling example:
;     LatchPos X, 1

PROC LatchPos Axis, Scode
STROBE [Axis] = Scode                  ; Select source for the strobe
LATCH Axis                             ; Initiate latching
REL [Axis] = 20000                     ; Move to relative coordinate
GO Axis
WAIT STA [Axis] & 0x0408               ; Wait for latching or end of motion
IF STA [Axis] & 0x0008
   HALT Axis                           ; Stop the motion
   PRINT "The latched coordinate is ", INDEX [Axis], #10
ELSE
   PRINT "No latched coordinate", #10
ENDIF
RETURN
```

AXIS HOMING

The homing procedure provides accurate initialization of the coordinate system of an axis. It is mandatory for every servo-controlled system. The home procedure should be executed before any motion is started.

The home procedure consist of the following basic steps:

- Set home speed and acceleration, set no position limits for the axis.
- Move to find where the home sensor changes its state.
- Move to latch the position of the first encoder index.
- Set the origin of the axis coordinate system.
- Restore position limits.

Here is an example of a home procedure:

```
; Procedure Name:
;        HomeAxis
;
; Parameters:
;        Axis        – name if the axis to be homed
;        HomeSensor  – name of the home sensor to be used
;        HomeOffset  – offset for the origin of the axis zero position
;
; Calling example:
;     HomeAxis X, InHomeX, 2500

; Golbal variables:
VAR SensorState                 ; Keeps initial state of the home sensor
VAR Fsave                       ; Temporary place for limit positions
VAR Rsave

PROC HomeAxis Axis, HomeSensor, HomeOffset

; Initial setup.
VEL [Axis] = 1000               ; Set home speed
ACC [Axis] = 2000               ; Set home acceleration

Fsave = FLIMIT [Axis]           ; Save current limit positions
Rsave = RLIMIT [Axis]
FLIMIT [Axis] = 0               ; Disable position limit check
RLIMIT [Axis] = 0
```

```
; Find home sensor edge.
IF IN [HomeSensor] == 0          ; Check for current state of the sensor
   REV Axis                      ; Request motion in reverse direction
ELSE
   FOR Axis                      ; Request motion in forward direction
ENDIF
GO Axis                          ; Start motion

; Wait until the home sensor change its state.
WAIT SensorState != IN [HomeSensor]
HOLD Axis                        ; Stop motion smoothly
WAIT STA [Axis] & 0x0400         ; Wait for motion to stop

; Latch the coordinate of the encoder index.
STROBE [Axis] = 14               ; Select encoder index to be the strobe
LATCH Axis
FOR Axis                         ; Allways search in forward direction
GO Axis
WAIT STA [Axis] & 0x0008         ; Wait until the position is captured
HALT Axis                        ; Stop motion smoothly
WAIT STA [Axis] & 0x0400         ; Wait for motion to stop
DELAY 1000                       ; Wait one second for motor to settle

; Initialize the axis position.
POS [Axis] = POS [Axis] – INDEX [Axis] - HomeOffset

; Restore position limits.
FLIMIT [Axis] = Fsave
RLIMIT [Axis] = RSave

RETURN
```

PARAMETERS SAVING AND RESTORING

Designing a motion control system requires some of the system parameters to be stored in the controller's non-volatile memory. The Logosol servo controller uses FLASH memory based disk emulator and the firmware supports the file oriented input / output operations.

The format in which the parameters are saved depends on the way they are going to be restored. Basically, there are two alternatives to completing this task:

- Interpreting of a "command" file by the command EXEC. This approach requires parameters to be stored as commands that if executed will set the given parameter. The advantage of this approach is that the parameter file is easy to read and the location of the parameters is not important. The disadvantage is that writing procedure is more complex and the parameter file is longer.

- Storing the data in a file as a sequence of numbers and parsing the file with the **_read** command. The advantage of this approach is that the parameter file is more compact, but the file is hard to read and the order of the parameters is fixed.

The example below illustrates how you could save speed and acceleration parameters of axis X:

```
; Name:         SaveParsX
;
; Parameters:   none
;
; Description: Save current speed/acceleration of axis X

PROC SaveParsX
_OPEN pars1.dat, w+                    ; Create a file called pars1.dat
_WRITE "VEL X = ", VEL X
_WRITE "ACC X = ", ACC X
_CLOSE                                 ; Close the parameter file
RETURN
```

The above example creates a file called PARS1.DAT with the following contents:

```
VEL X = 1000
ACC X = 2000
```

If the speed and acceleration are to be restored from this file the following command must be executed:

```
EXEC pars1.dat
```

Here is an example of storing the same parameters as a series of numbers (assuming it corresponds to a pre-defined format):

```
; Name: SaveParsX
;
; Parameters: none
;
; Description: Save current speed/acceleration of axis X

PROC SaveParsX
_OPEN pars1.dat, w+                    ; Create a file called pars1.dat
_WRITE VEL X
_WRITE ACC X
_CLOSE                                 ; Close the parameter file
RETURN
```

Here is how the parameters should be restored from this file:

```
; Name: RestoreParsX
;
; Parameters: none
;
; Description: Restore speed/acceleration of axis X

PROC RestoreParsX
_OPEN pars1.dat, r
_READ "%ld", VEL X
_READ "%ld", ACC X
_CLOSE
RETURN
```

> ***Note***
>
> *The length of all the parameter files stored in the FLASH-disk emulator should not exceed 40 Kbytes. If larger files have to be saved, contact Logosol about different FLASH-disk options.*

PONT-TO-POINT MOTION

The trajectory control commands specify the mode of motion (position, velocity or interpolation), target of the motion and also the beginning and end of the motion.

PROFILE	Specify trapezoidal or S-velocity profile
ARC	Request arc interpolation
LINE	Request line interpolation
ABSOLUTE	Request target for position mode
RELATIVE	Request relative target for position mode
GO	Start requested motion
VSPEED	Desired vector velocity
VACCEL	Desired vector acceleration
XAXIS	Axis identifier corresponding to the X coordinate
YAXIS	Axis identifier corresponding to the Y coordinate

POSITION-VELOCITY-TIME MOTION

The Position-Velocity-Time (PVT) mode of operation is a flexible mechanism for execution of complex trajectories. It is based on a set of points defining where and with what speed the axis should be at a given moment. Every two consecutive points define one segment of the desired trajectory. The PVT control algorithm calculates the axes velocity profile for every segment and takes care for the smooth transition from one segment to the next.

The definition of the PVT trajectory is described in a variable length structure. The header of this structure defines the number of the following fields and their type.

As the MCL language does not support complex data structures, the PVT definition is described below as number of cells from one-dimensional array with specific indexes.

The PVT definition structure has the following common format:

Header
Axes List
Segment List

The PVT header has the following format:

Index	Description
0	Number of axes to be synchronized (NumAxes)
1	Number of segments (NumSegments)
2	Round factor

The following list of axes is included after the header of the specified PVT trajectory (the number of the axes is defined at index 0 as NumAxes):

Index	Description
3	Index of the first axis
4	Index of the second axis
…	…
NumAxes + 2	Index of the last axis

The next list defines the target positions and the travel time for the first segment. This definition is followed by the same type of definition for the next segment and so forth. The number of segments to be described is defined at index 1 as NumSegments.

Index	Description
NumAxes + 3	Travel time
NumAxes + 4	Target position of the first axis
. . .	. . .
2 * NumAxes + 3	Target position of the last axis

SUPERVISOR MACRO IMPLEMENTATION

The Supervisor macro is intended to provide a mechanism for implementation of functionality similar to that of a PLC controllers. Unlike the normal execution of the macros in background or foreground tasks, the Supervisor macro is executed completely within a single time slice. It runs independently from the other tasks and could be used for implementation of control algorithms that run in parallel with the motion control procedures.

The supervisor macro is supposed to implement the functionality of a state machine. Every time slice it could check the inputs, the internal state and generate an output, which could be new digital output status and new internal state.

The firmware does not have a reserved macro name for "supervisor" macro. Instead, there is a command called **SUPERVISOR** that specifies the name of the macro to be executed every time slice. This way you could have as much supervisor macros as the number of internal states your state machine should have. This way the transition from one state to another is implemented by setting the corresponding "supervisor" macro.

The example below illustrates this technique with two-state machine. The transition from one state to another is controlled by the state of the input InSensor1. Setting the output RedLED indicates the current state of the machine.

```
PROC State1
IF IN [InSensor1] == 1
   SUPERVISOR State2
   OUT [RedLED] = 1
ENDIF
RETURN

PROC State2
IF IN [InSensor1] == 0
   SUPERVISOR State1
   OUT[RedLED] = 0
ENDIF
RETURN
```

> ***Note***
>
> *The supervisor task is non-preemptive and it will block the execution of any other tasks until the supervisor macro is processed completely. That is why the length of the supervisor macro should be as short as possible.*

The supervisor task could communicate with the other tasks through the global variables. It is also possible to start execution of a procedure in a background (**SUBMIT** command) from the supervisor task.

ERROR HANDLING

The Servo Controller firmware provides sophisticated methods for handling the different types of errors that might happen. They include Exceptions handling, Guard conditions and Internal Diagnostics.

Syntax Errors

The syntax errors are detected by the syntax analyzer during the processing of the system macros or when processing a single command. Misspelled commands, wrong parameters (numbers), a missing closing bracket, a missing match of **IF** and **ENDIF** statement, etc are considered syntax errors.

If a syntax error is detected in the system macro file, the processing of the file is interrupted and the program does not continue. If the syntax error is detected when a single command is entered, then the command is rejected and the prompt returned is a question mark (?), indicating a wrong syntax.

Motion Errors

The motion error occurs when the servo axis exceeds one or more of its limit parameters. The types of motion errors are:

- positive (forward) limit position violation
- negative (reverse) limit position violation
- maximum position (following) error exceeded
- maximum position error derivative exceeded
- position counter overflow

After a motion error is detected the normal execution of the procedure is interrupted and an Exception request is generated. The internal variable **EC** (Error Code) is set to indicate the corresponding reason for the Exception.

Run-time Macro Execution Errors

The run-time macro errors could happen when some of the variables calculated during the program execution are set to a value that is leading to an execution error. The types of run-time errors are:

- accessing an array cell out of the range (negative or too big index)
- dividing by zero

After a run-time error is detected the normal execution of the procedure is interrupted and an Exception request is generated. The internal variable **EC** (Error Code) is set to indicate that the reason for Exception is a run-time error.

Exceptions

Exceptions are abnormal events that happen during the operation of the machine. Possible exceptions in the Logosol servo controller are:

- Guard condition violation (described later)
- Motion error (limit position violation, maximum error, etc.)
- Hardware error (amplifiers overload)
- Emergency button pressed
- Run-time macro execution error

An exception macro is a special routine, which is executed only when the internal protection decides that a further execution of the normal operation is no longer possible. This is generally accompanied by terminating the motion. In this macro, the user has to take care of additional actions necessary for safely shutting down the system.

Guard Conditions

The **GUARD** statement installs a Boolean expression that is evaluated every time slice of the kernel. The expression could be a combination of all signals that must be in a definite state during normal operation. If one signal fails, the guard expression evaluates to false (0) and an exception event is generated.

The following example demonstrates a simple guard expression checking the state if an input – vacuum sensor:

```
GUARD IN [VacuumSensor]
```

If the Guard condition check is no longer needed, **GUARD** command should be executed without parameters.

Internal Diagnostic

The Internal Diagnostics provides information that is needed for the proper handling of an error or Exception. The access to the diagnostic data is implemented through the following variables:

- Error Line (**EL**) – holds the line number of the currently executed procedure at the moment an error is registered

- Error Axis (**EA**) – holds the index of the axis, which caused an error
- Error Time (**ET**) – holds the time when an error is detected
- Axis Status (**STATUS**) – indicates the reason of hardware protection activation (current overload, no power, shortage of an output)
- Error Reason Code (**RC**) – holds a code indicating the source of the Exception

In addition there is a circular buffer logging the macros invoked as well as the Exceptions. Every entry in this buffer has a time stamp. The content of the buffer is displayed with the **HISTORY** command. The contents of the buffer can be saved to a file if a file name is supplied as a parameter to the **HISTORY** command.

ABSOLUTE ENCODERS SUPPORT

The absolute encoders provide information about the position of an axis even after the power of the controller is turned off. Encoder operation without external power is supported by a backup battery up to one week following a power-down. In this “emergency power” mode the encoders are still keeping track of the axis position. Having the information from the absolute encoders the controller is capable of implementing sophisticated algorithms to recover from unexpected or emergency power-off situations. Such recovery algorithms are required in all cases when the homing of the machine is impossible if started from some particular configuration.

The supported absolute encoders are a combination of an incremental encoder and a multi-turn absolute encoder. The communication with the absolute encoder is through synchronous serial interface. It provides 32 bits of information for the current state of the encoder. The 24 bits return the current position of the encoder and the rest 8 bits are indicating the status of the absolute encoder.

The command returning the position of the absolute encoder is called **APOSITION**. The status is returned by the command **ASTATUS**. Look at the command reference for more details about the command syntax and the meaning of the status bits.

PID COEFFICIENTS OPTIMIZATION

The PID filter optimizer should be used exclusively during calibration or when considerable changes in machine mechanics or the payload are made. It is recommended to start the optimizer with relatively low current filter parameters, e.g. KP = 80, KD = 1000, KI = 80, IL = 70 and to perform no more than 10-15 iterations (see the parameter IterNo below).

Name: **OPT** - starts the PID-filter optimizer

Syntax:

OPT Axis [, Path, IterNo, Smpl, DefPars]

The arguments in square brackets are optional

Parameters:

Axis	the axis to be optimized
Path	the relative path to be executed
IterNo	number of iteration (usually greater than the number of cycles)
Smpl	number of samples to be recorded and used in calculating the criterion
DefPars	default initial parameter values. If this parameter presents, the optimization starts with the following values for the filter parameters: KP = 80, KD = 800, KI = 50 and IL = 50

> ***Note***
>
> *Path, IterNo, Smpl and DefPars are optional. The minimum parameter call of OPT is OPT Axis. The default values for the optional parameters are Path = 5000, IterNo = 10, Smpl = 255 and DefPars = _NA (Not Available).*

Example:

The following example optimizes the R axis parameters, performing 10 cycles of 10 inches relative motions, uses 255 samples to calculate the criterion starting from the current PID filter parameter set:

```
OPT R, 10000, 10, 255, 1
```

Macro Source Code:

```
PROC  OptProc Axis, Path, Smpl, IterNo, Cntr, MinC
PRINT "Number of samples: ", Smpl, #10
PRINT "N   KP  KI  KD  IL  Criterion", #10
RefPos = POS Axis
MinC = 999999999
bakK [0] = KP Axis
bakK [1] = KI Axis
bakK [2] = KD Axis
bakK [3] = IL Axis
optK [0] = KP Axis
optK [1] = KI Axis
optK [2] = KD Axis
optK [3] = IL Axis
IF IterNo < 0
   LoopNo = 0
   CntKP = 0
loop_KP:
   CntKD = CntKP
loop_KD:
   KP Axis = bakK [0] + CntKP * (RangeKP / 4)
   KD Axis = bakK [2] + CntKD * (RangeKD / 4)
   LoopNo = LoopNo + 1
   PRINT "Initial Parameter Set No. ", LoopNo, #10
ENDIF
KI Axis = bakK [1]
IL Axis = bakK [3]
Cntr = 0
TUNE -1
opt_loop:
REL Axis = Path
RECORD Smpl, _TS, Axis
GO Axis
WAIT !(STA [0] & 0x00200000) && (STA [0] & flgStop)
ABS Axis = RefPos
GO Axis
oldK [0] = KP Axis
oldK [1] = KI Axis
oldK [2] = KD Axis
oldK [3] = IL Axis
TUNE Axis                              ; call the optimizer
IF Criterion == -3
   WaitStop Axis
   GOTO opt_loop
ENDIF
Cntr = Cntr + 1
PRINT Cntr, ": ", oldK[0]," ", oldK[1], " ", oldK[2], " ", oldK[3], " : "
```

```
IF Criterion >= 0
   IF Criterion < MinC
      MinC = Criterion
      optK [0] = oldK [0]
      optK [1] = oldK [1]
      optK [2] = oldK [2]
      optK [3] = oldK [3]
   ENDIF
   PRINT Criterion, #10
ELSE
IF Criterion == -1
   PRINT MinC, " - Local optimum found!", #10
   GOTO end_label
ELSE
IF Criterion == -2
   PRINT "Limits violation ...", #10
   IF Cntr == 1
      GOTO end_label
   ENDIF
ELSE
IF Criterion = -4
   Print "Oscilations ...", #10
ENDIF
ENDIF
ENDIF
ENDIF
WaitStop Axis
IF IterNo < 0
   IF Cntr < 20
      GOTO opt_loop
   ENDIF
ELSE
   IF Cntr < IterNo
      GOTO opt_loop
   ENDIF
ENDIF

end_label:
PRINT "----------------------------", #10
IF IterNo < 0
   CntKD = CntKD + 1
   IF CntKD < 3
      GOTO loop_KD
   ENDIF
   CntKP = CntKP + 1
   IF CntKP < 3
      GOTO loop_KP
   ENDIF
ENDIF
```

```
KP Axis = optK [0]
KI Axis = optK [1]
KD Axis = optK [2]
IL Axis = 0
IL Axis = optK [3]
PRINT #10, "KP=",KP Axis, " KI=",KI Axis, " KD=",KD Axis, " IL=",IL Axis
PRINT "  : ", MinC, #10, "End of optimization", #10, #62
RETURN

Macro OPT Axis, Path, IterNo, Smpl, DefPars, AbsPath
; Parameters:
;        Axis        the axis to be tuned
;
;        Path        the relative path to be executed
;
;        IterNo      number of iteration (usually greater
;                    than the number of cycles)
;
;        Smpl        number of samples to be recorded and used
;                    in calculating the criterion;
;
;        DefPars     if present the optimization procedure
;                    starts with the following initial
;                    values of the PID-filter parameters:
;                    KP = 80, KD = 800, KI = 50, IL = 50
;                    otherwise the current parameters
;                    remain
;
;   Remark:     Path, IterNo, Smpl and DefPars are optional.
;               The minimum parameter call of OPT is OPT Axis.
;               The default values for the optional parameters are:
;               Path = 5000, IterNo = 10, Smpl = 255 and DefPars =_NA.

IF Path == _NA
   Path = 1000
ENDIF
AbsPath = Path
IF Path < 0
   AbsPath = - Path
ENDIF
IF IterNo == _NA
   IterNo = 35
ENDIF
SUBMIT  OptProc Axis, Path, 512, IterNo
RETURN
```

VIII. Motion Control Language Reference

This chapter describes the syntax and the function of all keywords – program statements, system commands and system variables of the MCL programming language.

COMMAND CATEGORIES

Commands Overview

The MCL programming is similar to using high-level language interpreters. The MCL source file is in ASCII text format. The file is loaded and preprocessed into a program buffer in binary codes. The MCL program is executed by the kernel, which translates the binary codes to LMC actions and flow control instructions.

Mainly, there are three groups of commands depending on their accessibility. The first group involves commands intended for system control: environment setup, loading and executing of MCL programs. These commands can be executed only from the command line. They are not available for MCL programs. One command category belongs to this group:

- System control

The second group consists of commands controlling program execution flow control and definition of MCL program items: variables, procedures, and macros. These commands are intended for use in programs only. They are not available from the command line. All commands of this group belong to the following category

- MCL program statements

The third group consists of commands interacting with the LMC boards. They are available either from the command line or from the MCL program. The following command categories belong to this group:

- Motion control parameters
- Run time parameters
- Interpolation parameters

- System parameters
- Motion commands
- Servo boards control
- Error handling
- File I/O functions

All command categories and commands that they involve are described below.

System Control Commands

The system control commands are intended for basic system operations like exit to DOS, invoking of file transfer utilities, redirection of command stream or getting help for commands. This command category includes the following statements:

QUIT	Stop motion and exit to DOS
DOS	Start a DOS shell
DOWNLOAD	Receive a file from remote computer
UPLOAD	Send a file to remote computer
VERSION	Display MCL version number

Program Control Statements

MCL program statements provide the necessary features for creating complex macro commands or implementing long algorithms. The program control statements are:

MACRO	Start new macro definition
PROCEDURE	Start new procedure definition
RETURN	Return to the calling macro or procedure
GOTO	Go to a specified label
WHILE - ENDWHILE	Loop with pre-condition
IF - ELSE - ENDIF	Conditional branch
WAIT	Wait until a condition is met
DELAY	Delay execution with the specified time

Motion Control Parameters

The motion parameters specify how the axes are to be moved. They should be set properly before starting a motion. The parameters specify the PID filter coefficients, the velocity and acceleration for the trapezoidal velocity profile and the limitations - the maximum position error, the maximum forward and reverse positions. All motion parameters could be changed during the motion except for the acceleration (the result will not take effect until the next move).

Commands are provided for saving and restoring motion parameters. They use an ASCII text to represent the parameter names and their values. Thus they could be edited easily.

The commands belonging to this group are:

VELOCITY	Desired velocity
ACCELERATION	Desired acceleration
DECELERATION	Desired deceleration
AJERK	Desired acceleration jerk
DJERK	Desired deceleration jerk
FLIMIT	Forward limit position
RLIMIT	Reverse limit position
MAXERROR	Maximum position error
DS	PID filter derivative sampling interval
KP	Proportional coefficient
KI	Integral coefficient
KD	Differential coefficient
IL	Integration limit

Current Axis Parameters

The run-time parameters describe the actual axis information. It alters during the axis motion. This information involves the current axis position, velocity, position error, status and index position. The commands reporting the run-time parameters are:

POSITION	Axis current (actual) position
INDEX	Axis index position
ERROR	Axis position error
TACHOMETER	Axis current (actual) velocity
STATUS	Axis and system information
APOSITION	Axis absolute encoder position
ASTATUS	Absolute encoder status
LINDEX	Axis latched index potision

It is important that the value returned by the Status command consists of two parts. The lower 16 bits contain axis-specific information and the upper 16 bits contain system specific information (common to all axes).

System Parameters

The System parameters specify the MCL behavior. They determine the command interpreter responses after processing a command. The response definition is controlled by a parameter with 20 bit-mapped options.

Other system-wide parameters determine scaling of the axis positions, i.e. translation of user defined units into encoder counts. There are two scaling coefficients that correspond to linear and rotational axis coordinates. These coefficients allow easy conversion from inches to millimeters and vice versa.

The following commands control the system parameters:

INFO	Specify response mode - verbal, numeric
REPORT	Define 20 bit-mapped response options
SCALE	Specify scaling coefficients
_EM	Exception event disable mask

Servo Boards Control Commands

The boards setup commands are intended for control of the LMC power supply, the servo loop, the digital inputs and outputs, the selection of an index source and the latching of an index position. The **RESET** command resets not only the LMC boards but the computer too. The Power command performs controller diagnostics and does not switch on the power if a hardware problem is detected.

SERVO and **NOSERVO** commands allow a single axis to be specified.

POWER	Switch on power supply to LMC boards
NOPOWER	Switch off power supply
SERVO	Close hardware servo loop
NOSERVO	Suspend servo control
IN	Get digital input state
OUT	Set/Get digital output
STROBE	Select index strobe source
LATCH	Enable latching of next index strobe
RESET	Reset servo boards and CPU

Error Handling Commands

The commands in this category are needed for proper handling of different kinds of errors that could occur during controller operation. The commands are:

EXCEPTION	Set the exception handling macro
_EM	Exception event disable mask
EL	Return the execution error line number
RC	Return reason code for an exception event
STOP/HALT	Stop motion and macro execution

File I/O Functions

The File I/O functions provide simple means for storing or retrieving calibration or configuration data from the controller FLASH based disk emulator. They could be used also for communication to other devices through the standard DOS device drivers.

The file-oriented functions are:

_OPEN	Open a file
_CLOSE	Close file
_READ	Read formatted data from a file
_WRITE	Write formatted data to a file
_PEEK	Returns the number of bytes in the input file buffer.

COMMANDS BY NAME

ABSOLUTE

Synopsis Defines axis target position.

ABSOLUTE *{axis = target}*
ABSOLUTE *{axis}*
variable = **ABSOLUTE** *axis*

axis	Defined name or index of an axis
target	Target position

Description The **ABSOLUTE** variable defines a new target position for a subsequent **GO** command.

The command requests position mode of motion and overwrites any motion requests defined by other commands as **RELATIVE, FORWARD, REVERSE, LINE** and **ARC**.

If more than one axis has absolute target position specified and the profile mode is 1 (see **PROFILE** parameter) the motion will be synchronized. That means if you have X/Y stage and specify absolute target for X and Y the stage will perform straight-line motion.

Example

```
>ABSOLUTE x=200 z=300 y=100
>ABSOLUTE
X=200
Y=100
Z=300
>GO
>
```

ACCELERATION

Synopsis Defines desired axis acceleration.

ACCELERATION *{axis = value}**
ACCELERATION *{axis}*
variable = **ACCELERATION** *axis*

axis	Defined name or index of an axis
value	Desired acceleration

Description The **ACCELERATION** is an axis variable defining desired accelerating and decelerating rate.

The acceleration must be greater than zero for a motion to start. If the desired acceleration is exceeding the resources of the mechanics and the controller then either the maximum position error will exceed its limit or the amplifiers current protection will shut down the power of the motors.

Example

```
>ACCELERATION x=1000 y=1500 z=300
>ACCELERATION
X=1000
Y=1500
Z=300
>xacc = ACCELERATION x
>xacc
1000
>
```

AJERK

Synopsis Defines the desired acceleration jerk for S-velocity profile.

AJERK *{axis = value}**
AJERK *{axis}*
variable = **AJERK** *axis*

axis	Defined name or index of an axis.
value	Define the acceleration jerk used for moving the axis.

Description The jerk is the first-time derivative of the acceleration. During motion using the S-velocity profile the acceleration jerk defines the smoothness of the acceleration profile.

The meaningful range of values for the acceleration jerk is from 0 to 1000. This parameter has no effect if trapezoidal velocity profile is selected.

Example

```
>AJERK x=1000 y=1500 z=300
>AJERK
X=1000
Y=1500
Z=300
>xajk = AJERK x
>xajk
1000
>
```

ARC

Synopsis Requests two axis coordinated motion on circular trajectory.

ARC *EndX, EndY, RdX, RdY, Dir**

EndX, EndY	End point of trajectory
RdX, RdY	Coordinates of circle center relative to current position
Dir	Direction of arc interpolation 0 – CCW, 1 – CW

Description Axes performing interpolation are determined by variables **XAXIS** and **YAXIS**.

VSPEED and **VACCEL** define the speed and acceleration during the interpolation mode of motion. These variables have to be set to the corresponding axes indexes that will perform **ARC** motion.

Motion requests specified by **ARC** and **LINE** commands are stored in a dedicated buffer that is intended for storing a complex motion trajectory definition, consisting of multiple lines and arcs. Execution of such motion trajectory is started with the **GO** statement.

ARC command overrides pervious motion requested by **ABSOLUTE**, **RELATIVE**, **FORWARD**, **REVERSE** , **LINE** or **PVT** commands.

Examples

```
>XAXIS = X
>YAXIS = Y
>VSPEED = 1000
>VACCEL = 500
>ARC 500, 1200, 100, 100, 0
>GO
>
```

APOSITION

Synopsis Returns the absolute encoder position.

APOSITION *{axis}*
variable = **APOSITION** *axis*

axis Defined name or index of an axis.

Description The variable returns the axis coordinate as preserved by an absolute encoder. See Absolute Encoder Application notes for more details about the use of **APOSITION** command.

Examples

```
>APOSITION X
X=120089
>
```

ASTATUS

Synopsis Returns the absolute encoder status.

ASTATUS *{axis}*
variable = **ASTATUS** *axis*

axis Defined name or index of an axis.

Description The **ASTATUS** variable returns the status of an absolute encoder. The data is returned as s decimal number. The meaning of the status bits is described in the following table:

Bit	Hex Mask	Description
0	00000001	Encoder read error
1	00000002	Encoder needs “pre-load”
2	00000004	Battery low
3	00000008	Encoder maximum speed exceeded
4	00000010	Encoder overflow
5	00000020	Battery error
6	00000040	Checksum error
7	00000080	Absolute encoder not installed

See Absolute Encoder programming examples for more details about the use of **ASTATUS** command.

Examples

```
>ASTATUS X Y
X = 0
Y = 0
>
```

BIAS

Synopsis Defines the motor output bias.

BIAS *{axis=value}*
BIAS *{axis}*
variable = **BIAS** *axis*

axis	Defined name or index of an axis.
value	Motor output bias

Description The **BIAS** variable sets the PID filter DC bias value. It is used to offset the constant unidirectional forces (typically a vertical axis which is not balanced by a counter-weight). The specified motor bias value is added directly to the output of the servo filter.

The motor bias has a range of –32767 to 32767.

Examples

```
>BIAS Y=100 X=200
>BIAS X Y
X=200
Y=100
>
```

CONST

Synopsis Defines a global constant in a macro file.

CONST *{Identifier Value}*

Identifier	Any valid identifier
V*alue*	A number

Description Constants are 32 bits signed integer numbers. They are useful to associate names with frequently used numbers or flags.

Examples

```
CONST inch 254
CONST ArmLength 10500
CONST FullTurn 36000
```

CLIMIT

Synopsis Defines the current limit for motor amplifier.

CLIMIT *{axis=value}*
CLIMIT *{axis}*
variable = **CLIMIT** *axis*

axis	Defined name or index of an axis.
value	Current limit

Description This parameter sets the current limit of the servo amplifiers. If this limit is exceeded for more than a 100 ms the amplifier power will be turned off.

The current limit of the servo amplifiers can be set in the range of 2 ÷ 10 A with 4-bit precision. The values are dimensionless quantities in the range zero to fifteen. The value of 0 sets the current limit to the lowest level (less than 1 A). The value of 16 sets the current limit to the highest level – 10 A.

Examples

```
>CLIMIT X
X = 8
>
```

DECELERATION

Synopsis Defines the deceleration for S-velocity profile.

DECELERATION *{axis = value}*
DECELERATION *{axis}*
variable = **DECELERATION** *axis*

axis	Defined name or index of an axis
value	Deceleration value

Description The **DECELERATION** variable specifies the desired maximum deceleration to be applied. This parameter is used only when S-velocity profile is set.

DECELERATION units are specified as distance / sec^2.

Example

```
>DECELERATION x=1000 y=1500 z=300
>DECELERATION
X=1000
Y=1500
Z=300
>xdec = DECELERATION x
>xdec
1000
>
```

DEFINE

Synopsis Defines expression.

DEFINE *name expression*

name	Unique identifier
expression	Expression definition

Description **DEFINE** allows you to associate unique name for an expression that is used frequently. After the name is defined it could be used instead of the expression.

The defined name is treated as a read only variable.

Examples

```
DEFINE AxisMoving ((Status X & $0400) == 0)
```

DELAY

Synopsis Delays execution of a macro.

DELAY *time*

time Delay time in milliseconds

Description The **DELAY** statement defines the further execution of a procedure. The accuracy of the requested delay depends on the specified time slice for the motion control kernel.

Examples The following example alters an output state every second:

```
PROC AlterOut OutID
again:
OUT [OutID] = ! OUT [OutID]
DELAY 1000                          ; Delay execution for 1 second
GOTO again                          ; Endless loop
RETURN
```

DJERK

Synopsis Defines the deceleration jerk for S-velocity profile.

DJERK *{axis = value}*
DJERK *{axis}*
variable = **DJERK** *axis*

axis	Defined name or index of an axis.
value	Deceleration jerk

Description The jerk is the first-time derivative of the deceleration. During motion using the S-velocity profile the deceleration jerk defines the smoothness of the deceleration profile.

The meaningful range of values for the deceleration jerk is from 0 to 1000. This parameter has no effect if trapezoidal velocity profile is selected.

Example

```
>DJERK x=1000 y=1500 z=300
>DJERK
X=1000
Y=1500
Z=300
>xjk = DJERK x
>xjk
1000
>
```

DOWNLOAD

Synopsis Starts external file transfer protocol to download a file.

DOWNLOAD [*filename*]

filename Name of the file to download

Description The **DOWNLOAD** command provides interface to a file transfer utilities.

The command starts file transfer program specified in the [SYSTEM] section of the configuration file, following *Download* keyword. The name of the file to be transferred is send as a command line parameter.

Examples

```
DOWNLOAD WhatCame.Dat
```

DS

Synopsis

Defines the PID filter derivative sampling interval in microseconds.

DS *{axis = sampling_interval}*
DS *{axis}*
variable = **DS** *axis*

axis	Defined name or index of an axis.
sampling_interval	Derivative sampling interval in microseconds

Description

The **DS** parameter may be in the range between 100 and 65536. It defines the time interval in microseconds between two adjacent samples of the encoder feedback to be used by the motion control processor.

The actual sampling interval set depends on the capabilities of the servo control processor installed. If the requested time interval is not available the closest available value will be set.

Examples

```
>DS x=256 y=512
>DS x y
X=256
Y=512
>
```

EA

Synopsis Returns the index of the axis caused exception.

Description If an axis motion stops for any unexpected reason, or limit position or maximum position error is violated the index of the axis caused error will be retained in the **EA** variable. **EA** is read-only variable.

Examples

>**EA**
2
>

EC

Synopsis Returns the code of the last command error.

Description The **EC** variable contains an error code that could be helpful for identifying the cause for the stop of the motion or an interrupt of the macro execution. **EC** is read-only variable. The table below lists the error codes:

Code	Description
0	No Error
10	Illegal motion control driver parameters
11	Motion control driver time out
12	Missing or bad servo board
13	Unable to turn on servo amplifiers power
30	Label cannot be declared
31	Bad label format
32	Label already defined
33	Unknown command
34	Command allowed only in procedure
35	Maximum number of commands exceeded
36	Bad command parameters
37	Identifier redefinition
38	Line overflow
39	Illegal command syntax
40	Procedure definitions cannot be nested
41	Procedure definition ends without RETURN
42	Maximum number of nested IF statements exceeded
43	No correspondent IF statement
44	Missing ENDIF statement

Code	Description
45	Maximum number of nested WHILE statements exceeded
46	No correspondent WHILE statement
47	Missing ENDWHILE statement
50	Unable to open file
51	Unable to read file
52	Insufficient memory to process file
60	Insufficient memory
61	STDIO operation
62	File I/O operation
63	Serial channel operation
64	Module reentrance determined
65	No open file for closing
69	Non existing procedure line number
72	Procedure execution not started
73	Line number required
75	File name required
76	Object required (Axes/procedures/Inputs/Outputs/Constants)
77	Sample number required
78	Command not allowed in resident mode
80	Invalid stack frame
81	Stack overflow
82	Stack empty
83	Go to undefined label
84	Motion control driver error
85	Invalid command index
86	Expression calculating
87	Jump out of range

Code	Description
88	Command not allowed during procedure execution
90	Controller has no power
91	Servo loop not closed
100	Index value out of range
101	Setting axis variable
102	Getting axis variable
103	Divide by zero
104	Internal expression error
105	Input cannot be set
110	Unknown expression component
111	Unknown identifier
112	No closing index bracket -]
113	No closing expression bracket -)
114	Invalid index expression
115	Invalid constant
116	Invalid operator usage
117	Too long identifier

Examples

```
>EC
65
>
```

EL

Synopsis Returns the line number where procedure execution has stopped.

Description If a procedure stops for any reason, the line number of the most recently processed command in the currently executed procedure file will be retained in the internal **EL** variable. **EL** is read-only variable.

Examples

```
>EL
217
>
```

ENCODER

Synopsis Returns the number of encoder counts per revolution.

ENCODER *{axis}*
variable = **ENCODER** *axis*

axis Defined name or index of an axis.

Description The **ENCODER** variable returns the number of encoder counts per revolution. This value represents the setting defined during axis description in the configuration (INI) file.

Examples

```
>ENCODER
X=2500
Y=2500
Z=2500
>
```

ERROR

Synopsis Returns the current position error.

ERROR *{axis}*
variable = **ERROR** *axis*

axis Defined name or index of an axis.

Description The **ERROR** variable returns the most recent position error determined by the servo control processor. It is equal to the difference between the current position and the desired position.

Examples

```
>ERROR
X=54
Y=30
Z=18
>
```

ET

Synopsis Returns the error time.

Description If a procedure stops abnormally for any reason, the internal timer value is stored in the **ET** variable. The value of **ET** is the number of timer ticks of the internal clock passed since the start of the MCL software.

Examples

```
>ET
217452
>
```

EXCEPTION

Synopsis Installs a macro as user defined exception macro.

EXCEPTION [*MacroName*]

MacroName Macro name

Description The macro specified by the **EXCEPTION** command is set as an error handler. The motion control kernel starts it upon detection of a critical event.

The following events cause the activation of the exception macro:

- Software limit position violation (**FLIMIT** and **RLIMIT**)
- Amplifiers power turned off
- Guard condition evaluates to false
- Exceeded maximum position error

The exception macro is intended to take precautions to activate actions that might be necessary to prevent damage to the complete system. Apart from other actions the host computer should be informed correctly and a working process of the master system might be stopped.

If no macro name is specified the exception routine is replaced by a default macro with the following contents:

```
MACRO DefaultException
      HALT
RETURN
```

Examples

```
MACRO RescueMe
      HALT
      NOSERVO
      NOPOWER
      ; ... more statements ...
RETURN

>EXCEPTION RescueMe
```

EXEC

Synopsis Executes commands from a text file.

EXEC [*FileName*]

FileName Name of the file to be used

Description The command **EXEC** opens a file and process every line as a command entered from the console of the host computer. This command could be used if you want to restore a set of parameters from a file. The executed macros are NOT stored in the RAM for further use.

Examples

```
>EXEC params.dat
>
```

FLIMIT

Synopsis Defines the positive limit position.

FLIMIT *{axis = limit}*
FLIMIT *{axis}*
variable = **FLIMIT** *axis*

axis	Defined name or index of an axis.
limit	Limit position or **NOLIMIT**

Description The **FLIMIT** variable is an axis parameter, specifying the maximum position in forward direction the axis is allowed to go to. If this limit is exceeded the kernel generates an exception and the motion is stopped. This variable is intended to provide software limit of the range of motion for an axis.

There is a predefined constant **NOLIMIT** that deactivates limit position check. It is equal to zero.

Examples

```
>FLIMIT x=12000 y=360000 z=7000
>FLI
X=12000
Y=360000
Z=7000
>
```

FORWARD

Synopsis Requests jogging forward.

FORWARD *{axis}*

axis Defined name or index of an axis.

speed Specify the velocity to be used to move the axis.

Description The command requests axis motion in velocity mode (also called jogging). Motion starts after the execution of a **GO** command.

The command overwrites any motion requests defined by other motion commands like **RELATIVE**, **ABSOLUTE**, **REVERSE**, **LINE** or **ARC**.

Examples

```
>POS X
1800
>VEL X=1000
>FOR X
>FOR Y
GO                          ; Both X and Y should move now
>POS X
2000
>POS X
2760
>HALT X Y                   ; Neither X nor Y will move
>
```

GO

Synopsis Starts motion.

GO *{axis}*

axis Defined name or index of an axis.

Description The command **GO** starts a motion. Any of the commands **ABSOLUTE, RELATIVE, FORWARD, REVERSE, PVT, LINE** or **ARC** request the motion that will be performed, but no action is taken until the motion is started with a subsequent **GO** command.

The type of axis motion is determined by the last issued motion request command:

Command	Motion mode requested
Absolute/Relative	Position mode
Forward/Reverse	Velocity mode (jogging)
Line/Arc	Interpolation mode
PVT	PVT Mode

If no axis is specified as a parameter for the **GO** command all axes having motion requests are started to their respective targets.

Examples

```
>FOR X
>POS X
3000
>POS X
3000
; nothing happened; now start motion with Go
>GO X
; wait again for a little while, axis should move meanwhile
>POS X
3500
>
```

GOTO

Synopsis Unconditional jump to the specified label.

GOTO *label*

label Any label appearing in the current procedure

Description **GOTO** is a program statement executing unconditional jump to a predefined label. The labels are local for every procedure.

Examples

```
Loop1:                          ; label definition in separate line
ABS X=1000                      ; any command
. . .                           ; more commands
GOTO Loop1                      ; jump to the label
```

GUARD

Synopsis Installs expression as user defined exception condition.

GUARD [*expression*]

expression Expression that evaluates to false, if the system is in a critical situation.

Description The **GUARD** command is intended to provide user-defined condition to be verified every time slice. If such a condition occurs the normal execution of the macro is interrupted and the exception macro is started.

There is a dedicated flag in the **STATUS** word to reflect **GUARD** violation.

The **GUARD** expression is meant to define a logical expression that describes the normal (uncritical) state of a selected set of signals, usually a set of input signals.

If no expression is specified, evaluation of expression is terminated.

Examples

```
GUARD IN[VacValve] == 0
; The above expression will initiate exception if the vacuum
; valve input detects that the vacuum is off.
```

HALT

Synopsis Stops motion with the programmed deceleration.

HALT *{axis}*

axis Defined name or index of an axis.

Description The **HALT** command stops the axis smoothly, with the programmed deceleration. If no axis is specified then all axes are stopped and any procedure being executed is terminated.

If you need to stop an axis abruptly in an emergency situation, use the **STOP** command.

Note that you should not use the **HALT** command in a procedure without parameters. If you do so the **HALT** command will be the last one executed from the procedure – it will terminate the procedure execution.

Examples

```
>HALT X                    ; decelerate axis X to a standstill
>HALT                      ; Stop procedure execution and any motion
>
```

HISTORY

Synopsis

Shows the history of the invoked macros and exceptions.

HISTORY *{HistoryLines| FileName}*

FileName	Valid DOS filename to open
HistoryLines	Number of history line to be returned

Description

The **HISTORY** command displays the 50 most recently started macros or exceptions.

The recorded statements are displayed in order of execution.

If no *HistoryLine* is specified all samples are displayed continuously starting with the line following the last displayed line.

If a *FileName* is specified then the history is stored to the file. If a file with the same name exists it is overridden

Examples

```
>HISTORY
Prompt [81] STA   Timer=16.422
>
```

IF - THEN - ELSE

Synopsis Provides conditional branch in the macro.

IF *expression*

{statements}

[**ELSE**

{statements}]

ENDIF

expression An expression that evaluates to 0 if and only if the condition is false

Description The **IF - ELSE - ENDIF** statements provide standard mechanism for conditional branch. The number of the nested **IF** statements should not exceed 80.

Examples

```
IF IN VacuumSwitch == 1
      InsertObject
ELSE
      InformOperator
ENDIF
```

IL

Synopsis

Defines the integral limit for the motion servo processor.

IL *{axis}*
IL *{axis =integral_limit}*

axis	Defined name or index of an axis.
integral_limit	Number between 0 and 32000

Description

The **IL** limits the restoring force of the corresponding axis performed by the integral term (**KI**). The integral limit defines an upper boundary, which the integral sum may grow to.

The complete set of motion parameters (**KP, KI, KD, IL** and **DS**) have to be in harmony to control the overall behavior of the PID regulators of the servo control processor. Choosing wrong values for the motion parameters may result in a strange behavior of the axis such as jitter, vibrations or no motion at all.

Examples

```
>IL y=60 x=80
>IL x y
X=80
Y=60
>
```

IN

Synopsis Returns the current state of an input.

IN *{inputname}*

inputname Predefined input name or expression that evaluates to a valid input number

Description The **IN** variable returns the state of a predefined input. It must be one of the defined inputs in the INI file. You could use either the name of the input or its index.

If the command is used without any parameter at all, the actual definitions for all defined inputs are reported.

Examples

```
>IN VacuumSwitch
1
>xSwitch = IN VacuumSwitch
>xSwitch
1
>
```

INDEX

Synopsis Returns the value of the index register.

INDEX *{axis}*

axis Defined name or index of an axis.

Description The **INDEX** variable returns the contents of the index register corresponding to the specified axis. This register latches the current position of the axis upon the rising or the falling edge of strobe signal selected. Refer to the **STROBE** command for details about the selection of the strobe signals.

The command **LATCH** initiates the latching procedure. It has to be executed first in order to enable the operation of the hardware.

Examples

```
DEFINE FlgIndex 0x0008
PROC GetIndex
LATCH X
REVERSE X
GO
WAIT (STATUS(X) & FlgIndex)
STOP X
RETURN INDEX X
```

INFO

Synopsis

Sets the notification mode.

INFO *informationmode*

Informationmode	**NONE**	Respond only on request
	DEC	Respond as a decimal number
	HEX	Respond as a hexadecimal number
	TEXT	Respond as an ASCII string

Description

The **INFO** command selects how you'd like to be notified about the asynchronous events and if you want to have the name of axes displayed in a front of the axis parameter requested.

If **INFO TEXT** is chosen, the status is decoded into a series of text strings showing the status as a readable text. In addition when you request an axis variable value, the name of the axis is displayed in a front followed by equal sign. If everything is correct i.e. a zero status is returned, the corresponding text message will be OK.

Examples

```
>INFO HEX
>
0B40
>INFO NONE
>
>
>INFO TEXT
>
Procedure processor state: RUNNING
Executed procedure line 114
System timer contents 55123
System power on
System servo on
>
```

KD

Synopsis Defines the differential coefficient KD.

KD *{axis = diff_coeficient}*
KD *{axis}*
variable = **KD** *axis*

axis	Defined name or index of an axis.
diff_coefficient	Differential coefficient for the motion servo processor

Description The **KD** coefficient influences the smoothness of motion of the corresponding axis.

KD provides a force that is proportional to the rate of change of the position error. Correct settings contribute to an appropriate damping resulting in less perturbation.

The complete set of motion parameters (**KP, KI, KD, IL** and **DS**) have to be in harmony to control the overall behavior of the PID regulators of the servo control processor. Choosing wrong values for the motion parameters may result in a strange behavior of the axis such as jitter, vibrations or no motion at all.

Examples

```
>KD X=2000
>KD X
X=2000
>
```

KI

Synopsis

Defines the integral coefficient KI.

KI *{axis = integral_coefficient}*
KI *{axis}*
variable = **KI** *axis*

axis	Defined name or index of an axis.
integral_coefficient	Integral coefficient for motion servo processor

Description

The **KI** coefficient provides a restoring force that grows with time and thus insures that the static position error becomes and remains zero.

The complete set of motion parameters (**KP, KI, KD, IL** and **DS**) have to be in harmony to control the overall behavior of the PID regulators of the servo control processor. Choosing wrong values for the motion parameters may result in a strange behavior of the axis such as jitter, vibrations or no motion at all.

Examples

```
>KI y=1000 x=2000
>KI x (x+1) y
X=2000
Y=1000
Y=1000
>
```

KP

Synopsis Defines the proportional coefficient KP.

KP *{axis = prop_coefficient}*
KP *{axis}*
variable = **KP** *axis*

axis	Defined name or index of an axis.
prop_coefficient	Proportional coefficient for the motion servo processor

Description The **KP** coefficient provides a restoring force proportional to the position error.

KP is used each time the PID regulators detect a deviation between the expected target and the actual position.

The complete set of motion parameters **(KP, KI, KD, IL** and **DS**) have to be in harmony to control the overall behavior of the PID regulators of the servo control processor. Choosing wrong values for the motion parameters may result in a strange behavior of the axis such as jitter, vibrations or no motion at all.

Examples

```
>KP y=1000 x=2000
>KP x (x+1) y
X=2000
Y=1000
Y=1000
>
```

KPHASE

Synopsis Defines the velocity phase advance gain.

KPHASE *{axis=value}*
KPHASE *{axis}*
variable = **KPHASE** *axis*

axis	Defined name or index of an axis.
value	Velocity phase advance gain.

Description The **KPHASE** variable specifies the velocity phase advance gain. The value specified has an allowed range of 0 to 32767. This parameter can be changed on the fly if desired.

This variable is meaningful for brushless servo control boards only.

Examples

```
>KPHASE X = 100
>KPHASE X
X=100
>
```

LATCH

Synopsis Enables position latching of the next index strobe.

LATCH *{axis}*

axis Defined name or index of an axis.

Description The command **LATCH** is intended to setup the hardware for latching the current position of the specified axis into a dedicated index register. The index register is accessible by the **INDEX** variable. The source of the strobe signal to activate the latching is specified by the **STROBE** variable.

Example

```
DEFINE FlgIndex 0x0008
PROC GetIndex
LATCH X
REVERSE X
GO
WAIT (STATUS(X) FlgIndex)
HALT X
RETURN INDEX X
```

LINE

Synopsis Requests linear interpolated motion.

LINE *EndX, EndY*

EndX, EndY End point X/Y positions.

Description Axes performing interpolation are determined by variables **XAXIS** and **YAXIS**. These two variables have to be set with the indexes of the corresponding axes.

The speed, acceleration and maximum position error during the coordinated motion are defined by **VSPEED** and **VACCEL** variables.

Motion requests specified by Arc and Line commands are stored in a dedicated buffer. It is intended to store a complex motion trajectory definition, consisting of multiple lines and arcs. Execution of such a motion trajectory is started with a **Go** statement.

The **LINE** command overrides the previous motion requested by the **ABSOLUTE**, **RELATIVE**, **FORWARD**, **REVERSE**, **ARC** or **PVT** commands.

Examples

```
>XAXIS = X
>YAXIS = Y
>VSPEED = 1000
>VACCEL = 500
>LINE 5000, 2000
>GO
>
```

LIST

Synopsis Shows the source code of the currently loaded macro.

LIST *[ProcOrMacro | LineNumber][, Count]*

ProcOrMacro	Name of the procedure or macro to be displayed
LineNumber	First line number to be displayed
Count	Number of lines to be displayed

Description The **LIST** command displays the source code of the requested procedure, macro or a program fragment starting with the specified line number.

If no value for count is specified, 20 lines will be displayed.

Examples

```
>LIST Startup     ; list the first 20 lines from macro Startup
>LIST 125         ; list 20 lines starting from  line 125
>LIST 125,10      ; list 10 lines starting from line 125
```

MACRO

Synopsis Defines macro.

MACRO *macroname[,parameter {,parameter}]*

macroname	Valid and unique identifier
parameter	Valid and unique identifier

Description The **MACRO** statement defines a macro name and its parameters. The macro definition ends with a **RETURN** statement.

The maximum number of parameters is 10. Every parameter is considered optional. If a parameter is not supplied a reserved constant (**_NA**) is set to it. Later in the macro you could find if a parameter has valid setting by comparing its value with **_NA.**

Examples

```
; Sample macro with two parameters.
MACRO RCP Axis1 Axis2
IF Axis1 == _NA || Axis2 == _NA
      PRINT POS X, " " POS Y
ELSE
      PRINT POS Axis1, " ", POS Axis2
RETURN

; Calling the macro.
>RCP X Y
2200
12000
>
```

MAXERROR

Synopsis Defines the maximum allowable position error.

MAXERROR *{axis=value}*
MAXERROR *{axis}*
variable = **MAXERROR** *axis*

axis	Defined name or index of an axis.
value	Position error

Description The variable specifies the maximum units which are tolerable as a position error by the servo processor. If the position error is greater than the value set with **MAXERROR**, the user procedure will be stopped and the exception macro will be called.

The maximum value allowed is 32767 encoder counts. Setting of **MAXERROR** to 0 will disable checking of position error.

Examples

```
>MAXERROR Y=1000 X=2000
>MAX X (X+1) Y
X=2000
Y=1000
Y=1000
```

MESPEED

Synopsis Specify the maximum allowable position error derivative.

MESPEED *{axis=value}*
MESPEED *{axis}*
variable = **MESPEED** *axis*

axis	Defined name or index of an axis.
value	Maximum error speed.

Description The **MESPEED** variable specifies the maximum position error derivative. The control of this parameter helps for reducing the impact caused by a mechanism hitting an obstacle.

Examples

>**MESPEED** X=1000
>**MESPEED** X
X=1000
>

MLIMIT

Synopsis Defines the maximum motor power from the servo board.

MLIMIT *{axis=value}*
MLIMIT *{axis}*
variable = **MLIMIT** *axis*

axis	Defined name or index of an axis.
value	Maximum current.

Description The **MLIMIT** variable sets the maximum allowed motor command value output from the PID filter. The command is useful when you want to avoid mechanical damage caused by excessive power delivered from the servo board.

The range of **MLIMIT** is from 0 to 32767. If the magnitude of the PID filter output value (whether positive or negative) exceeds the motor limit then the output value is maintained at the motor limit value. Once the filter output value returns below the specified limit then normal servo filter values are output.

Examples

```
>MLIMIT Y=1000
>MLIMIT Y
Y=1000
>
```

MOTOR

Synopsis Defines the direct output value supplied to the motor.

MOTOR *{axis=value}*
MOTOR *{axis}*
variable = **MOTOR** *axis*

axis	Defined name or index of an axis.
value	Motor output.

Description The **MOTOR** variable specifies an output value applied to the motor power amplifier when the servo is turned off. The allowed range is from –32767 to 32767. The value of 0 indicates no motor power will be applied.

Examples

```
>MOTOR Y=1000
>MOTOR Y
Y=1000
>
```

NAME

Synopsis Shows the names of the specified objects.

NAME *{DisplayObject}*

DisplayObject		A group of items out of the following:
	AXES	List all axes names
	PROC	List all procedure names in memory
	MACRO	List all macro names in memory
	IN	List all input names
	OUT	List all output names
	VAR	List all names of defined variables
	CONST	List all names of defined constants
	DEFINE	List all expressions defined with Define

Description The command allows you to obtain the names of the defined objects in the servo controller. This command is useful to avoid the name conflicts.

If no *DisplayObject* is specified, all objects are displayed.

Examples

```
>NAME Proc          ; lists defined procedures
>
```

NOPOWER

Synopsis Turns off the power.

NOPOWER

no parameters

Description The command **NOPOWER** turns off the power of the servo board. All outputs are switched to their inactive state. The high voltage power relay is opened to disconnect the controller from the external power supply.

The power to the encoders is retained after **NOPOWER** command, so you don't have to "home" the system next time the power is turned on.

Note that this command will turn off the servo amplifiers as well.

Examples

```
>NOPOWER
0K
>
```

NOSERVO

Synopsis Turns off the servo amplifiers.

NOSERVO *{axis}*

axis Axis name or index.

Description The command **NOSERVO** turns off the servo loop for the specified axis. Current axis position remains valid because the encoder power is not turned off.

If a brake is associated with the axis, its power will be turned off as well in order to hold the axis in its position and to prevent the mechanism from falling down.

If no axes are specified the servo control of all defined axes will be turned off.

Examples

>**NOSERVO**
0K
>

OUT

Synopsis Controls outputs.

OUT *{outputname = outputstate}*
OUT *{outputname}*

outputname	Predefined output name or index
outputstate	Output value

Description All outputs are defined in the configuration profile. They have name and active state.

The name could be used as a reference. The active state specifies how to be set an activated output.

Examples

```
>OUT sw1=1 sw2=0
>OUT sw1 sw2
1
0
>
```

POSITION

Synopsis Returns the current position.

POSITION *{axis = newPositionValue}*
POSITION *{axis}*
variable = **POSITION** *axis*

axis	Defined name or index of an axis.
newPositionValue	Any integer expression that evaluates to a valid axis position
variable	Any variable declared with **VAR** statement

Description The **POSITION** variable returns the current position of the axis.

Assignment of a value to the **POSITION** variable overwrites the internal position setting with the new value. This results effectively in a transformation of the axes coordinate system. **POSITION** shifts the origin of the coordinate system to the assigned position. All subsequent references to an axis position are relative to the reference point defined with **POSITION**.

The values are internally stored as encoder counts and all calculations are done in units of encoder counts. Yet, the transformation like every assignment and retrieval of the contents of a variable is done in units that are defined by the **SCALE** command.

Examples

```
>POSITION Y
Y=342
>POSITION x=0 z=0
>Y POSITION=180000
>POSITION
X=0
Y=180000
Z=0
>
```

POWER

Synopsis Turns on the main power.

POWER

Description The **POWER** command turns on the motor power relay and starts a diagnostic sequence. All outputs are set to the initial state defined in the configuration (.INI) file. The motion processor(s) is initialized and the amplifiers are enabled.

Diagnostic procedure requires all axis positions to not change for 200 ms. If some axes move the power is shut down

Examples

```
>POWER
>
```

PRINT

Synopsis Prints data to the terminal.

PRINT *{value|"text"|#char}*

value	Any valid expression
"text"	String to be print
char	ASCII code of character to be print

Description **PRINT** lets you display any value to the console. This enables a programmer to provide information to the console operator. The argument **PRINT** command could be an expression, string or ASCII code.

Examples

```
>PRINT "Positions:", #10, #13, Pos X, #10, #13, Pos Y
Positions:
2200
12000
>
```

PROC

Synopsis Declares procedure and its parameters.

PROC *procname [parameter {parameter}]*

Procname	Any valid and unique identifier
Parameter	Any valid and unique identifier

Description The **PROC** statement declares procedure name and parameters. The maximum number of parameters is 10. All parameters are considered optional. If no parameter is specified at the procedure invocation a dedicated constant (_**NA**) is assigned to the parameters

The procedure definition is terminated with the **RETURN** statement.

Examples

```
; Example of procedure TEST with three parameters
PROC TEST Axis, Speed, Accel
VEL Axis = Speed
ACC Axis = Accel
RETURN

>TEST X, 2000,1000            ; invoking procedure TEST
>X VEL ACC
2000
1000
>
```

PROFILE

Synopsis Defines the velocity profile.

PROFILE *{axis = newValue}*
PROFILE *{axis}*
variable = **PROFILE** *axis*

axis	Defined name or index of an axis.
newValue	An integer expression: 0 Trapezoidal 1 S-Velocity profile 2 Spline velocity profile
variable	Any variable declared with **VAR** statement

Description The **PROFILE** command defines the velocity profile of an axis. When S-profile is specified the axis variables **ACCELERATION**, **DECELERATION**, **AJERK**, **DJERK** and **VELOCITY** are used to define the motion trajectory.

Examples

```
>PROFILE Y
Y=0
>PROFILE x=0 z=0
>Y PROFILE = 1
>PROFILE
X=0
Y=1
Z=0
```

PVT

Synopsis Prepare for motion in Position-Velocity-Time mode.

PVT *pvt_array*

pvt_array An array containing the PVT points definition

Description The command **PVT** requests motion in PVT mode defined by the points descriptions stored in the *pvt_array*

See also “Position-Velocity-Time Motion” on page 45.

Examples

```
>PVT PointsA[0]
>GO
```

RC

Synopsis Returns the reason code for procedure termination.

Description If a procedure execution has stopped **RC** returns a code for the reason of the termination. The codes are listed in the following table.

0	No procedure started or execution terminated normally
4	**STOP** Command executed
5	**HALT** Command executed
6	**RETURN** statement executed
7	User defined limit position reached
8	Error in current statement
9	Error in Supervisor macro processing
10	Error in Exception macro processing
11	Maximum position error exceeded
12	Bad hardware response
13	Wraparound in hardware counters
14	Error in math library
15	Error when processing Guard expression
16	Guard condition violated
17	Time slice overrun

Examples

```
>RC
6
```

RECORD

Synopsis Records axis positions.

RECORD *Samples TimeDelta {axis}*

Samples	Total number of samples to be taken.
TimeDelta	Time between two subsequent samples [ms]
axis	Defined name or index of an axis.

Description There are a limited number of samples of the actual axis position recorded. The maximum number of taken samples is defined in configuration file, [SYSTEM] section, by the "Samples" keyword.

The time between two samples is defined by the *TimeDelta* parameter. The smallest time is equal to the motion kernel time slice.

If no axis is specified the samples are taken for all axes known to the system.

See also **SAMPLES**

Examples

```
>RECORD 5 10  X
>SAMPLES
970147      149
972083      148
974013      148
975940      196
977863      197
```

RELATIVE

Synopsis Defines relative target position.

RELATIVE *{axis = distance}*
RELATIVE *{axis}*
variable =**RELATIVE** *axis*

axis	Defined name or index to an existing axis.
distance	Any valid expression .

Description The distance is used to calculate the new axis target for a subsequent **Go** command using the trapezoidal velocity profile. New target is equal to sum of distance and current axis position.

The command requests position mode of motion and overwrites any other motion request commands like **ABSOLUTE**, **FORWARD**, **REVERSE**, **LINE**, **ARC**, or **PVT**.

Examples

```
>POSITION
X=1000
Y=1000
Z=1000
>RELATIVE x=200 z=300 y=100
>GO X Y Z
>POSITION
X=1200
Y=1100
Z=1300
```

REPORT

Synopsis Defines the report format.

REPORT [[=] *SelectionMask]*

SelectionMask An integer number that is a bit representation of the selected display items

Description When the system is in an interactive verbal mode, there is a number of items that can be reported each time a blank line is sent from the console. Report chooses those items.

Setting the corresponding bit to 1 does the selection.

Bit	Hex Mask	Description
0	00000001	Procedure processor state (reports: IDLE or RUNNING)
1	00000002	Currently executed procedure line number
2	00000004	Procedure stop reason (if procedure is stopped)
3	00000008	Procedure stop error (if stopped bacause of error)
4	00000010	Procedure stop time
5	00000020	Procedure stop line
6	00000040	System time in timer ticks
7	00000080	Supervisor procedure name
8	00000100	Exception procedure name
9	00000200	Guard expression
10	00000400	Record statement execution status
11	00000800	System power state
12	00001000	Hardware servo loop state

Bit	Hex Mask	Description
13	00002000	User command processing error
14	00004000	User command processing acknowledge
15	00008000	User procedure call acknowledge
16	00010000	Notify on procedure stop
17	00020000	Notify on axis motion stop
18	00040000	Reserved
19	00080000	An axis is moving
20	00100000	Servo OFF on some axis

If **REPORT** command is specified without a parameter, it returns the active setting of the report word.

Examples

```
>REPORT 0x0001
>
```

RESET

Synopsis Resets the servo controller.

RESET

no parameters

Description The controller performs cold reboot.

The controller is reset to its power up state. All I/O lines are set to the hardware initial values.

Actually, Reset does the following:

- Stops the motion of all axes
- Executes the **NOSERVO** statement
- Executes the **NOPOWER** statement
- Resets the motion control processor internal logic
- Initiate a cold boot of the PC system

The power up settings of the I/O lines are determined by the settings defined in the configuration file. Those are initialized with the **POWER** command only.

Examples

>**RESET**
>

RETURN

Synopsis Defines the end of macro or procedure.

Synopsis **RETURN** *[expression]*

expression Any value to be returned as a result of procedure execution

Description The **RETURN** statement defines the end of procedure or macro. You could have more than one **RETURN** statement. Optionally you could return a value to the calling procedure by entering an expression after the statement. The return value is stored in the variable **RV**. If no expression is specified then zero is returned by default.

Examples

```
; Example of procedure TEST with three parameters
PROC TEST Axis, Speed, Accel
VEL Axis = Speed
ACC Axis = Accel
RETURN

>TEST X, 2000,1000            ; invoking procedure TEST
>X VEL ACC
2000
1000
>
```

REVERSE

Synopsis Jog axis backwards using the predefined velocity and acceleration.

REVERSE *{ axis }*

axis Defined name or index of an axis.

Description The command **REVERSE** requests motion in velocity mode (also called jogging). Motion starts after **GO** command is issued.

The command overwrites any other motion request commands like **RELATIVE**, **ABSOLUTE**, **FORWARD**, **LINE, ARC** or **PVT**.

Examples

```
>POS X
13800
>REV X
>GO X
>POS X
12000
>POS X
7560
>
```

RLIMIT

Synopsis

Define the minimum allowed position for an axis target.

RLIMIT *{axis = limit }*
RLIMIT *{axis }*
variable = **RLIMIT** *axis*

axis	Defined name or index of an axis.
limit	Any integer expression that evaluates to a valid axis position or **NOLIMIT**

Description

The **RLIMIT** command is intended to limit the working envelope of an axis. If the axis is commanded to go below the defined limit the motion will be stopped and the exception macro will be started. The axis status word indicates such position limit violation with a dedicated flag.

If the **RLIMIT** is set to 0 then the position limit violation is ignored.

Examples

```
>RLIMIT x=-12000 y=-100 z=0
>RLI
X=-12000
Y=-100
Z=0
>
```

RV

Synopsis Returns the value set by the Return statement of the calling procedure.

Description Every procedure returns a value that is passed as an argument to the **RETURN** statement. The returned value is stored in the **RV** variable. Then, it could be processed by the calling routine.

Examples

```
MACRO TestRV
RETURN 129

>TestRV
>RV
129
>
```

SAMPLES

Synopsis Lists the position samples recorded with the RECORD command.

SAMPLES *[SampleNumber | FileName]*

SampleNumber Number of samples to be returned

Description The command **SAMPLES** returns a list of the position and error samples recorded by the **RECORD** command. Sample number 1 refers to the first recorded sample of the last **RECORD** command.

If no sample number is specified all samples are displayed. If a file name is specified instead of a number then the samples are stored in the file in ASCII format.

A sample has the following format:

[Position][Space][Position Error][CR][LF]

See also **RECORD**

Examples

```
>RECORD 5 10 X
>SAMPLES
970147 149
972083 148
974013 148
975940 196
977863 197
```

SCALE

Synopsis

Defines the global scaling factors.

SCALE *[LinearScaling] [,RotationalScaling]*

LinearScaling	Scaling factor for linear motions
RotationalScaling	Scaling factor for rotational motions

Description

Converting of the user defined units to encoder counts and vice versa is scaled by coefficients defined by the **SCALE** command. It provides individual scaling for the linear and for the rotational axes. Setting the scaling factors to zero eliminates all conversions, i.e. all measurement units will be in encoder counts.

This command allows switching between different units measurement. Thus, if system is set up to accept millimeters as distance unit, defining of scaling factor of 0.001 will enable the specification of distances in microns. Respectively, a scaling of 0,0393 ($=25.4^{-1}$) would allow specifying of inches instead of millimeters.

Internally, all distances are transformed into encoder counts. Although the scaling factor defines the precision used, the accuracy cannot be higher than the resolution of the installed encoders.

There are two different scaling factors available. The linear scaling factor is used for axes that are defined as linear axes with the L parameter in the axis setup of the [SYSTEM] section of the initialization file. The rotational scaling factor is used for axes that are defined as rotational axes with the R parameter in the axis setup of the [SYSTEM] section of the initialization file.

If neither linear nor rotational scaling is specified, current settings of scale factors are returned to console.

Examples

```
>SCALE 2, 0.1
>SCALE
2.000000  0.100000
>
```

SERVO

Synopsis Activates the servo loop and release brakes.

SERVO *{axis}*

axis Defined name or index of an axis.

Description The command **SERVO** turns on the motor amplifiers power and activates the servo loop controlled by the servo processor. If a brake is associated with the axis it is released automatically.

After closing the servo loop the axis will feel stiff when trying to move it manually. If the axis is forced to move from its target position by an external force, the servo loop will break as soon as the deviation between target position and actual position becomes greater than the limit defined by the **MAXERROR** variable.

If no axes are specified the servo loop for all axes known to the system is closed.

Examples

```
>SERV0
0K
>
```

STATUS

Synopsis Returns the current status of the system and axis.

STATUS *[axis]*
axis Defined name or index of an axis.

Description Return the current system status encoded as a 32 bit integer value.

- The upper 16 bits represent the current system status. They are common for all axes.
- The lower 16 bits represent the current status of the specified axis.

The data is always reported in hexadecimal format regardless of the INFO setting.

Bit #	Hex Mask	Description
0	00000001	Acquire next index pulse
1	00000002	Forward limit (**FLIMIT**) violated
2	00000004	Reverse limit (**RLIMIT**) violated
3	00000008	Index pulse acquired
4	00000010	Undefined
5	00000020	Undefined
6	00000040	Interpolator is operating
7	00000080	Axis servo off
8	00000100	Emergency input activated
9	00000200	Axis turned off because of excessive position error
10	00000400	Target position reached
11	00000800	Moving in velocity mode

12	00001000	Moving on forward direction
13	00002000	Amplifier current overload
14	00004000	Output shortage or operating power is missing
15	00008000	Board power failure
16	00010000	User command error
17	00020000	User procedure execution error
18	00040000	Supervisor execution error
19	00080000	Servo loop error (broken)
20	00100000	Interpolation in action
21	00200000	Recording in action
22	00400000	User procedure is executing
23	00800000	Guard condition is violated
24	01000000	Supervisor is installed
25	02000000	Exception macro is installed
26	04000000	Reserved
27	08000000	Reserved
28	10000000	Reserved
29	20000000	Reserved
30	40000000	System is not initialized
31	80000000	System has no power

STOP

Synopsis Stops motion of an axis immediately.

STOP *{axis}*

axis Defined name or index of an axis.

Description The axis is stopped immediately with the highest possible deceleration.

If no axis is specified all axes are stopped. In addition, if the axis parameter is not specified, the execution of any running procedure is interrupted.

Examples

>**ST0P**
0K
>

STROBE

Synopsis Selects the source of the reference strobe signal.

STROBE *{axis = strobesource}*

axis	Defined name or index of an axis.
strobesource	Source of index strobe according to the table below

Description This variable is used by the **LATCH** command to determine the source of the strobe signal. This signal is intended to detect the reference position of the axis. Strobe values select the input line according to the hardware spec of the controller.

> ***Note***
>
> *The strobe source codes vary between different servo control boards. Verify that data with the hardware documentation of the board you are using*

The strobe input is common for all axes assigned to the same board. Thus, changing of the strobe source for a single axis will reflect on the other two axes controlled from the same board.

Axes indexes 0, 1, 2 and 3 correspond to numbers, used in the axis name definition in the configuration file. Default strobe code is 14.

Examples

```
>STROBE X = 14
>STROBE X
14
```

SUBMIT

Synopsis Starts procedure or macro execution in the Background task.

SUBMIT *macroname [parameters]*

macroname	Macro or procedure name
parameter	Macro parameter list

Description If a macro (defined by **MACRO** command) is called from the terminal, it is executed completely (in the Foreground task) and then control is returned to the calling program. **SUBMIT**, however, starts the specified macro execution in the Background task, to be executed concurrently with the other tasks of the motion control kernel.

Only one background task could be running at a given moment of time. If the Submit command is invoked while the Background task is busy, it will return an error.

Examples

```
; Definition of the macro Move
MACRO Move axis target
      ABS axis=target
      GO axis
      WAIT STA [axis] & 0x400
      PRINT "*** Move: moved to desired target", #10, #13
RETURN

>SUBMIT Move X 5000     ; submitting the macro to the background
>X POSITION
X=2200
>X POSITION
X=4000
*** Move: moved to desired target
>X POSITION
X=5000
>Move X 1000            ; executing the macro in foreground
*** Move: moved to desired target
>X POSITION
X=1000
```

SUPERVISOR

Synopsis Installs supervisor macro.

SUPERVISOR *[MacroName]*

Description The specified macro is installed and executed completely every time slice.

If no macro name is specified the supervisor is deactivated.

Examples

```
MACRO Watchout
IF IN [Sensor1] & IN [Sensor2]
      OUT Valve1=1
ELSE
      OUT Valve2=1
ENDIF
RETURN

>SUPERVISOR WatchOut
OK
```

TACHOMETER

Synopsis Returns the current velocity of an axis.

TACHOMETER *{axis}*
variable = **TACHOMETER** *axis*

axis Defined name or index of an axis.

Description **TACHOMETER** command returns the current axis velocity. It is calculated every time slice based on the position samples.

Examples

```
>TACHOMETER X Z
X=1223
Z=3000
```

TIMER

Synopsis Returns or sets the current value of the MCL countdown timer.

TIMER *[[=] period]*

period Time period in milliseconds to count down

Description The **TIMER** command returns the current value of a countdown timer. The initial value of the timer if set by assigning a new value to the *period* parameter.

Examples

```
>TIMER = 1000
OK
>TIMER
670
>TIMER
221
>TIMER
0
>TIMER
0
>
```

UPLOAD

Synopsis Starts file transfer to upload a file to the host.

UPLOAD *filename*

filename Any string, containing the DOS path name of the file to be uploaded

Description The **UPLOAD** command invokes the file transfer program specified in the [SYSTEM] section of the configuration file.

Examples

```
>UPLOAD motion.mcl
>
```

VACCEL

Synopsis Defines the desired velocity for coordinated motion – vector acceleration.

VACCEL *[= accel]*

accel Desired vector acceleration.

Description The **VACCEL** sets vector acceleration used during coordinated motion requested by **LINE** or **ARC** statements.

Examples

>**VACCEL** = 400
>**VACCEL**
400
>

VAR

Synopsis Defines global variables.

VAR *{variablename}*

variablename Any valid identifier

Description Variables can be defined within macro files. They are 32 bits signed integer numbers.

Array dimension is defined by enclosing the array size in square brackets. Only one-dimensional arrays are allowed.

Examples

```
VAR xVelo
VAR vArray[10]
VAR V1, V2, V3
VAR S = 10              ; Default value
```

VELOCITY

Synopsis Defines the desired velocity.

VELOCITY *{axis = speed}*
VELOCITY *{axis}*
variable = **VELOCITY** *axis*

axis	Defined name or index of an axis.
speed	Desired velocity

Description Specifies the maximum desired velocity.

Examples

```
>VELOCITY x=2000 z=3000 y=1000
>VELOCITY
X=2000
Y=1000
Z=3000
>
```

VERSION

Synopsis Returns the version number of the firmware.

Description The command returns the kernel version number and copyright information.

Examples

```
>VERSION
Motion Control Language Version 2.43
(c) Copyright 1993,1999 Logosol, Inc.
```

VSPEED

Synopsis Defines the desired vector velocity for coordinated motion.

VSPEED *[= speed]*

speed Expression that evaluates to valid vector velocity.

Description Vector speed is used when **LINE** or **ARC** statements request coordinated motion.

Examples

```
>VSPEED = 1000
>VSPEED
1000
>
```

WAIT

Synopsis Delays the macro execution.

WAIT *expression*

expression An arithmetic or boolean expression representing the condition to wait for

Description The further execution of the procedure is delayed until the expression evaluates to a non-zero value, i.e. until the condition becomes true.

Examples

```
DEFINE AxisMoving ((Status x & $0400) == 0)
ABSOLUTE X=1000
GO X
WAIT !AxisMoving
```

WHILE - ENDWHILE

Synopsis Specifies a loop.

WHILE *expression*
ENDWHILE

expression An expression that evaluates to 0 if and only if the condition is false

Description If the expression is evaluated to FALSE the further execution of the procedure continues with the statement after **ENDWHILE**.

If the expression is evaluated to TRUE then procedure execution continues with the statement after **WHILE**. The corresponding **ENDWHILE** statement will return the procedure execution to the **WHILE** statement.

Examples

```
PROC Line4
RV = 0
WHILE RV << 4
      PRINT "Line ", RV, #10
      RV = RV + 1
ENDWHILE
RETURN

>Line4
Line 0
Line 1
Line 2
Line 3
>
```

XAXIS

Synopsis Defines X axis to be used in coordinated motion.

XAXIS *[= axis]*

axis Defined name or index of an axis.

Description The specified axis is used when **LINE** or **ARC** commands request coordinated motion.

Examples

```
>XAXIS = X
>XAxis
0
>XAXIS = Y
>XAxis
1
>
```

YAXIS

Synopsis Define Y axis to be used in coordinated motion.

YAXIS *[= axis]*

axis Defined name or index of an axis.

Description The specified axis is used when **LINE** or **ARC** commands request coordinated motion.

Examples

```
>YAXIS = Y
>YAxis
1
>XAXIS -= X
>XAxis
0
>
```

_CLOSE

Synopsis Closes a previously open file.

Description The **_CLOSE** command closes a previously open file. Only one file could be open at any given time. Thus, an open file should closed, before opening another one.

Examples

```
>_OPEN "test.txt", "r"
>_CLOSE
>
```

_OPEN

Synopsis Opens a file for I/O operation.

_**OPEN** *[filename, mode]*

filename Name of the file to open
mode Mode of file operation - read, write or both.

Description The command **_OPEN** is intended to enable read or write operations to a file. The file could be open in one of the following modes:

Mode	Description
r	Open in read-only mode.
r+	Open for update (r/W)
w	Opens file for reading and writing. The file must exist.
w+	Opens file for reading and writing. If the file does not exist it is created.

If the **_OPEN** command fails an error code is stored in the **_IOERROR** variable.

Examples

```
>_OPEN test.dat, w+
>_WRITE “test data”
>_CLOSE
>
```

_READ

Synopsis

Reads formatted information from a file.

***_READ** [format_string, variable]*

format_string C style formatting string.
variable Variable to read data into.

Description

The **_READ** command allows to enter information from a file into macro variables. This is implemented by using the format string describing the format of the data. The firmware processes the string and knows how to convert the data read from the file.

The string could contain one of the following formatting characters:

Format string	Type of the data to be expected from the file
"%ld"	Long (32 bit) integer
"%c"	Character

The file you want to read from has to be open before the **_READ** command is used.

If the command fails an error code is stored in the **_IOERROR** variable.

Examples

```
PROC RestoreParsX
_OPEN pars1.dat, r
_READ "%ld", VEL X
_READ "%ld", ACC X
_CLOSE
RETURN
```

_WRITE

Synopsis

Writes formatted information to a file.

WRITE *[format_string, variables]*

format_string C style formatting string.
variable Variable to be written into file.

Description

The **_WRITE** command writes a formatted string to a file. This is implemented by using the format string describing the format of the data. The firmware processes the string and knows how to convert the data to be written to the file.

The string could contain one of the following formatting characters:

Format string	Type of the data to be expected from the file
"%ld"	Long (32 bit) integer
"%c"	Character

The file must be open before the use of **_WRITE** function.

If the command fails an error code is stored in the **_IOERROR** variable.

Examples

```
>_WRITE "Vel X = %ld", VEL X
>
```

_PEEK

Synopsis Returns the number of bytes in the input file buffer.

_PEEK *[result]*

result Variable to hold the result

Description The function returns the number of bytes in the input file buffer.

The file must be open before the use of **_PEEK** command.

Examples

```
>_OPEN test, r
>_PEEK temp
>_READ "%ld", VEL X
>_CLOSE
>
```

APPLICATION NOTES

XTTY.SYS DOS DEVICE DRIVER

Synopsis DEVICE=XTTY.SYS Com:[Baud[,Parity[,Data[,Stop]]]]

Description

Com (COM1 | COM2)
Serial port to be used for console redirection.

Baud (12 | 24 | 48 | 96 | 19 | 38 | 115)
Baud rate for the serial port: The exact baud rate or the first two digits of the baud rate can be specified. If nothing or a blank is specified at the position for the baud rate, the system's actual baud rate remains unmodified.

Parity (NONE | ODD | EVEN)
Specifies the Parity for the serial port. Only the first character of the parity may be specified. If nothing or a blank is specified at the position, the system's actual parity remains unmodified.

Data (7 | 8)
Specifies the Data bits for the serial port. If nothing or a blank is specified at the position, the system's actual data bits setting remains unmodified.

Stop (1 | 2)
Specifies the Stopbits for the serial port.
If no parameter is specified, the current setting remains unmodified.

Installation Install the communication driver in CONFIG.SYS

In order to reroute the console I/O from standard DOS CON to a terminal connected to the serial port, it is necessary to install the provided communication driver XTTY.SYS as a DOS device driver in CONFIG.SYS.

Redirection is performed automatically by MCL if XTTY driver is installed. The driver could be used also as a parameter of the DOS command CTTY issued from the DOS prompt.

Example Installing the communication driver to COM2

```
DEVICE=C:\MCL\XTTY.SYS COM2:96,n,8,1
```

MCL.EXE MOTION CONTROL KERNEL

Synopsis MCL [-p *ConfigFile*] [*SystemMacroFile*] [*UserMacroFile*]

Description

SystemMacroFile	Name of an existing DOS file that contains system macros.
UserMacroFile	Name of an existing DOS file that contains user defined macros.
ConfigFile	Name of a configuration file to be used instead of MCL.INI.

When you invoke MCL.EXE and no *ConfigFile* is specified, MCL searches for configuration file named MCL.INI. Although it is generally located in the same directory as the MCL.EXE file, it can also be in some other directory. The current directory and the home directory of MCL.EXE are searched for it automatically. If *ConfigFile* is not found, MCL.EXE terminates with an error message.

There are no default filename extensions associated with the files.

Logosol, Inc.
1155 Tasman Drive
Sunnyvale, CA 94089
USA

Tel: (408) 744-0974
Fax: (408) 744-0977
http://www.logosolinc.com

Document No 715009001

Printed in USA