Preface

Chapter 1 – GML Commander Overview

Chapter 2 – Configuring Control Options

Chapter 3 – Configuring Axis Use

Chapter 4 – Using the Imaginary Axis

Chapter 5 – Working with Blocks

Chapter 6 – Working with Diagrams

Chapter 7 – Working With Modules

Chapter 8 – Working with Program Scripts

Chapter 9 – Control Setting Blocks

Chapter 10 – Motion Blocks

Chapter 11 – I/O and Event Blocks

Chapter 12 – Program Control Blocks

Chapter 13 – Multitasking Blocks

Chapter 14 – Status Blocks

Chapter 15 – AxisLink Blocks

Chapter 16 – RIO Blocks

Chapter 17 – CNET Blocks

Chapter 18 – DH-485 Blocks

Chapter 19 – SLC Blocks

Chapter 20 – Calculation Blocks

Chapter 21 – Display and Operator Interface Blocks

Chapter 22 – Miscellaneous Blocks

Chapter 23 – Defining Variables, Constants, and I/O

Chapter 24 – User Variables and Constants

Chapter 25 – The Expression Builder

Chapter 26 – Motion Variables

Chapter 27 – Controller Variables

Chapter 28 – Fault Variables

Chapter 29 – Status Variables

Chapter 30 – Diagnostic Variables

Chapter 31 – System Functions

Chapter 32 – Mathematical Functions

Chapter 33 – Using AxisLink 655

Chapter 34 – Using the RIO Adapter Option

Chapter 35 – Configuring ControlNet (CNET) 753

Chapter 36 – 1394 GMC Turbo SLC Interface

Chapter 37 – Initializing Your Motion Controller

Chapter 38 – Axis Locked and Axis Done Conditions

Chapter 39 – Merging Different Motion Types

Chapter 40 – Going Online

Appendix C – ASCII Reference

INDEX

Preface

Read this preface to familiarize yourself with this manual. This preface covers the following topics.

- Prerequisite knowledge
- Who should use this manual
- Purpose of this manual
- Contents of this manual
- Where to find help
- Using the optional Novice Mode
- Related documentation
- Rockwell Automation support
- About the GML Commander software application
- Common techniques used in this manual

Prerequisite Knowledge

To use GML Commander, you should be familiar with the operation of Microsoft Windows95 or Windows NT 4.0. You should also understand how to use your motion controller.

For more information, see your *Microsoft Windows95 User's Guide* or *Window NT 4.0 User's Guide.* In addition, refer to the installation and setup manual for your motion controller.

Who Should Use this Manual

Use this manual if you are responsible for designing, testing, or debugging GML Commander™ diagrams used with Allen Bradley® controllers. GML Commander is a member of the GML™ software product family.

Purpose of this Manual

This manual is a reference manual for the GML Commander programming tool. It contains explanations of all the features of this software. It describes the procedures you use to design a diagram, define the parameters of each function, download, test, and debug the diagram, all using the graphical interface of GML Commander.

Where to Find Help

GML Commander provides four types of help:

- A Reference Manual
- A Trouble Shooting Manual
- Online help
- Novice Mode

Using the Manual Set

This manual is part of the documentation set for GML Commander:

Use this manual:	Publication Number:	To find information on this topic:
GML Commander CD Insert	GMLC-5.3.CDI	Installing the software Starting the software Defining the hardware for GML Setting up the system properties and the required configurations
GML Commander Reference Manual	GMLC-5.2	Setup details Expression Builder details Block function details
GML Commander Troubleshooting Manual	GMLC-5.4	Error messages Faults

Using Online Help

Several types of online help are available:

To use this:	Do this:	Description:
GML Commander help	Select **GML Help Topics** from the Help menu.	• Descriptions of menus, screens, blocks, variables, constants, and I/O • Configuring the controller • How-to information
Context help	Select the Help icon from the toolbar and drag it onto the item in the workspace and click with your left mouse button, or select the item in the workspace and press **F1**.	A description of the item

Using the Optional Novice Mode

If you have never used Windows-based software or if you feel that you need more assistance, we recommend Novice Mode. If you are in Novice mode, after you complete a dialog box, the next one automatically appears. Novice Mode also provides added information on some dialog boxes. A final dialog box informs you that the configuration task is complete.

In this manual we assume that Novice Mode is not selected. Normally, the tabs appear at the top for you to select additional dialog boxes. This convention assumes that after opening the initial dialog box that you:

- Select a tab to open additional dialog boxes.
- Select OK to accept entered values and close each dialog box.

Related Documentation

The following documents contain additional information concerning related Allen-Bradley products. To obtain a copy, contact your local Allen-Bradley office or distributor.

For:	Read this Document:	Publication Number:
Information regarding 1394 hardware installation and setup procedures	1394 Digital, AC, Multi-Axis Motion Control System User Manual	1394-5.0
Features of the 1394 family	1394 Digital, AC, Multi-Axis Motion Control System Brochure	1394-1.0
Specifications of 1394 family	1394 Digital, AC, Multi-Axis Motion Control System Product Data	1394-2.0
Information regarding IMC S Class Compact hardware installation and setup procedures	IMC-S/23x Compact Motion Controller Installation and Setup Manual	4100-999-122

Specifications of the IMC S Class Compact motion controllers	IMC S Class Compact Motion Controllers Product Data	4100-2.3
Information regarding IMC S Class Integrated hardware	IMC-S/21x Integrated Motion Controller Installation and Setup Manual	4100-999-103
Specifications of IMC S Class Integrated motion controllers	IMC S Class Integrated Motion Controllers Product Data	4100-2.1
Information regarding IMC S Class Basic hardware	IMC-S/20x Basic Motion Controller Installation and Setup Manual	4100-999-105
Specifications of IMC S Class Basic motion controllers	IMC S Class Basic Motion Controllers Product Data	4100-2.0
An article on wire sizes and types for grounding electrical equipment	National Electrical Code	Published by the National Fire Protection Association of Boston, MA
A complete listing of current Allen-Bradley documentation, including ordering instructions. Also indicates whether the documents are available on CD-ROM or in multiple languages	*Allen-Bradley Publication Index*	SD499
A glossary of industrial automation terms and abbreviations	*Allen-Bradley Industrial Automation Glossary*	AG-7.1

Contents of this Manual

Chapter	Title	Contents
Preface	*Preface*	This chapter provides an introduction to GML Commander. It describes what you need to know about GML Commander before you build your first diagram. It includes: • The different functional segments of GML Commander. • What function block palettes are available to make your diagram easy to build. • Menus and toolbars.
1	*Overview*	This chapter contains general procedures you use while creating and editing diagrams.
2	*Configuring Control Options*	This chapter describes the steps to define the control options configuration settings. It also contains additional step-by-step procedures that let you customize your workspace based on the controller you are using. It includes information about: • Remote IO. • SLC. • AxisLink. • DH-485. • Multidrop.
3	*Configuring Your Axes*	This chapter contains step-by-step procedures you use to define the axis configuration settings.
4	*Using the Imaginary Axis*	This chapter tells how to configure and use an imaginary axis.
5	*Working with Blocks*	This chapter covers the physical aspect of blocks, such as selecting, positioning, and manipulating. .
6	*Working with Diagrams*	This chapter contains step-by-step procedures you use to create, test, and document your diagram.
7	*Working with Modules*	This chapter includes procedures for creating, viewing, and documenting modules.
8	*Working with Scripts*	This chapter provides procedures for using the script editor and translating a diagram to script.

Chapter	Title	Contents
23	*Defining Variable, Constants, and I/O*	This chapter describes how to set optional variables, constants, and I/O values. It includes: • Displaying system tags. • Creating, editing, and deleting user variables, constants, inputs, and outputs. • Displaying, adding, and deleting tags from the Watch window.
24	*User Variables*	This chapter explains how to create user specific variables.
25	*Expression Builder*	This chapter explains use of the Expression Builder to create calculations.
26	*Motion Variables*	This chapter explains the System Variables that fall into the Motion category.
27	*Control Variables*	This chapter explains the System Variables that fall under the Control category.
28	*Fault Variables*	This chapter explains Fault Variables.
29	*Status Variables*	This chapter explains the meaning and use of the Status variables.
30	*Diagram Variables*	This chapter explains the System Variables that fall under the Diagram category.
31	*System Functions*	This chapter explains the use of System Functions.
32	*Mathematical Functions*	This chapter explains the use of Mathematical Functions.
33	*Using AxisLink*	This chapter explains how GML Commander deals with Virtual Axes, AxisLink I/O, and fault handling when using AxisLink.
34	*Using the RIO Adapter Option*	This chapter describes how to setup and use the Remote I/O with GML Commander.
35	*Configuring ControlNet (CNET)*	This chapter describes how to setup and configure ControlNet for use with GML Commander.
36	*1394 GMC Turbo SLC Interface*	This chapter walks you through the setup and use of the SLC Interface.

Chapter	Title	Contents
37	*Initializing Your Motion Controller*	this chapter tells you how to initialize the hardware and software for your motion controller.
38	*Axis Locked & Axis Done Conditions*	Axis Locked and Axis Done conditions are explained in this chapter.
39	*Merging Different Motion Types*	This chapter explains how to merge motion of different types.
40	*Going Online*	This chapter includes procedures for downloading a diagram, debugging and fine-tuning, and uploading controller options.
41	*CPU Utilization*	This chapter provides an understanding of how CPU time is utilized and how to make the most efficient use of the CPU resources.
42	*Memory Organization*	This chapter explains how your motion controller uses memory and how Commander uses the motion controller's memory.
Appendix A	*Data Parameters & Data Bits*	This appendix lists all of the data parameters and data bits for Commander along with their values, units, and defaults.
Appendix B	*Understanding Differences Between GML 3.x & Commander Function Blocks*	This appendix shows how some blocks function differently in Commander than they did in GML 3.x.
Appendix C	*ASCII Reference*	This appendix provides tables containing the ASCII codes and characters used by GML Commander.
	Index	An alphabetical listing of topics covered within this manual.

Rockwell Automation Support

Rockwell Automation offers support services worldwide.

Local Product Support

Contact your local Allen-Bradley representative for:

- Sales and order support
- Product technical training
- Warranty support
- Support service agreements

Technical Product Assistance

If you need technical assistance, first review the information in this manual. If you need more information, call your local Allen-Bradley representative.

For the quickest possible response, we recommend that you have the catalog numbers of your products available when you call. You should also have the software version and the control firmware version numbers available. See *Where to Find Help* in this chapter for the publication numbers related to this product.

The Rockwell Automation Technical Support number is:

1-603-443-5419

On the Web

For information about Allen-Bradley, visit the following World Wide Web site:

http://www.ab.com/

About the GML Commander Software Application

GML Commander uses the information that you provide to customize your screens. For example, if you tell it that you have only two axes. Any information that would normally pertain to a third or fourth axis would either not appear or be grayed out.

Techniques Used in this Manual

The following conventions are used throughout this manual:

- Bulleted lists provide information, not procedural steps.
- Numbered lists provide sequential steps.
- Words that you type or select and keys that you press appear in bold.
- Field names and references appear in italics.
- Warnings appear with the following symbol:

ATTENTION: This warning identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. This symbol indicates a situation that requires immediate attention for personnel safety or for preventing harm to machinery.

Important: Identifies information that is critical for successful application and understanding of the product.

- The term select means that you use your mouse cursor to point to the value and then click-and-release the left mouse button to mark your choice. Depending on the field, you can select one or more options for a field. The options appear in various formats:
    - Sometimes you must browse through a list to find the value you

want. Clicking an item in the list highlights your selection.

- Sometimes you can select several values in one area. A check in a box ☑ is used when you can select more than one option.
- Sometimes only one value is allowed. A radio button is used when you can select only one option.
- When you select a block in your diagram, the block is highlighted. You can now:
  - •Move the block by clicking and dragging, rather than clicking and releasing.
  - •Open a selected dialog box by double-clicking or pressing the Enter key.

In all cases, the term indicates your choice to GML Commander.

Chapter 1

GML Commander Overview

GML, the exclusive Graphical Motion Control Language from Allen-Bradley, provides a graphical method of programming your motion controller. This revolutionary tool reduces the time you need to learn motion control programming and makes problem-solving easier. GML integrates software programming and debugging to solve your motion control problems. GML Commander, a member of the GML family, is a Microsoft® Windows®-based interface to the following controllers:

- 1394 GMC (IMC S Class 1394)
- 1394 GMC Turbo (IMC S Class 1394 Turbo)
- IMC S Class Compact
- IMC S Class Basic
- IMC S Class Integrated
- 1394L GMC (IMC S Class 1394L)

Before you begin using GML Commander, read this chapter to become familiar with:

- How it works
- Understanding your workspace
- Understanding your tools

How it Works

GML Commander uses a natural, flowchart-oriented approach to motion control programming. You create a diagram of your application solution by placing function blocks representing actions on the screen and connecting them in the proper order to achieve the sequence of operations. You enter motion and process details later using a fill-in-the-form approach.

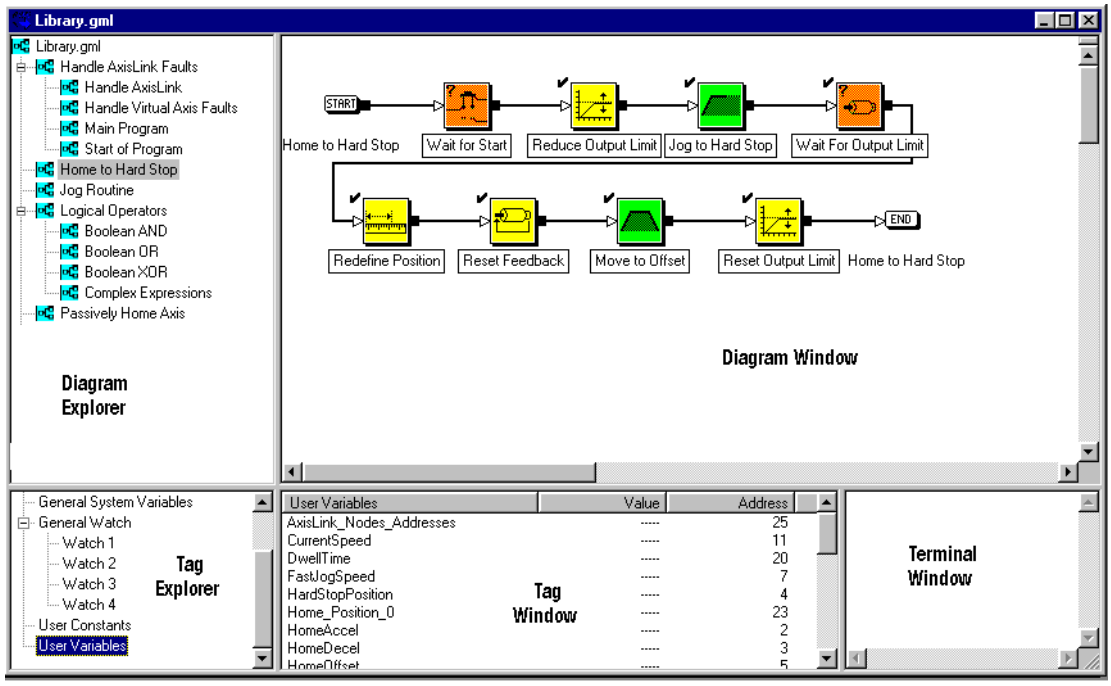
After the diagram is completed, you translate it into a script in the native language of the motion controller. You download this script to the motion controller. You need the PC for application program development. You do not need it for the final turnkey application.

When GML Commander opens for the first time, you see several work areas called views that allow you to perform different tasks. You can create or edit programs in a graphical environment and transparently translate them into the native language of the motion controller.

Understanding Your Workspace

GML Commander provides the workspace within which you execute your application programs. The workspace consists of five views. When you first open GML Commander, the workspace is relatively empty waiting for you to begin using the functions and features.

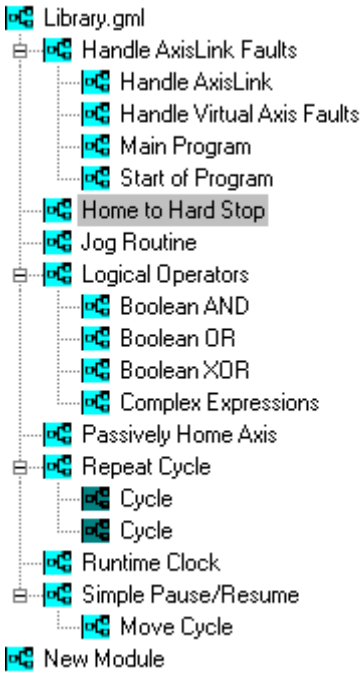
Titles have been added to the example below to identify the different functional areas:



This workspace provides all functions and features that you need to create, edit, and test diagrams. It also provides access to all menus and toolbars.

Understanding the Diagram Explorer

The Diagram Explorer provides a hierarchical tree that shows the organization of information.



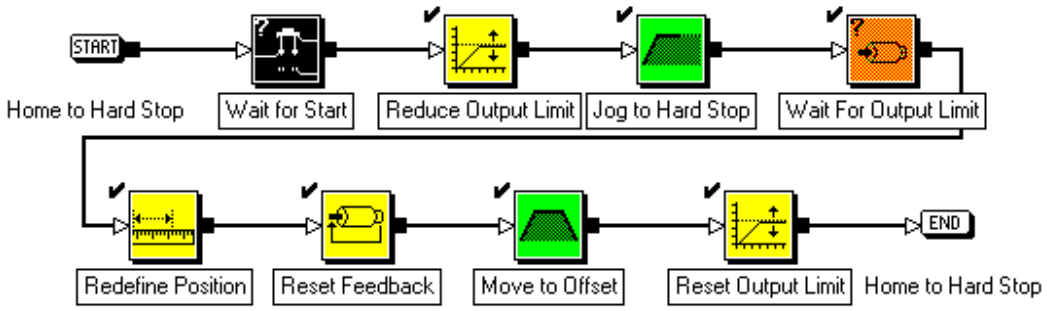
GML Commander uses the same methods that Microsoft Windows Explorer uses to move, open, and view the blocks and modules within a diagram. You can:

- Navigate or edit programs on a module level.
- Explode or collapse branches.
- Drag and drop a module from the tree's node onto another tree's node. The new module retains not only its tags, blocks and connections, but also its branches.

The Diagram Explorer shows the contents of a diagram. When you select a module from the Diagram Explorer, blocks and connections appear in the Diagram Window.

Understanding the Diagram Window

The Diagram Window provides a graphical interface to manage diagram development and maintenance, as shown in the example below:

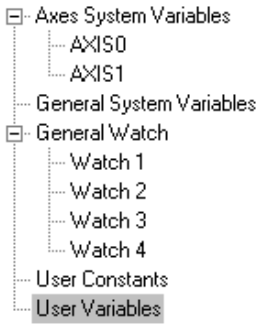


A new Diagram Window is active each time you open GML Commander and each time you select a new diagram. Though several Diagram Windows can be open simultaneously, only one is active at one time.

Note: Each diagram has a start and end block at the root level with one or more functional blocks connected together.

Understanding the Tag Explorer

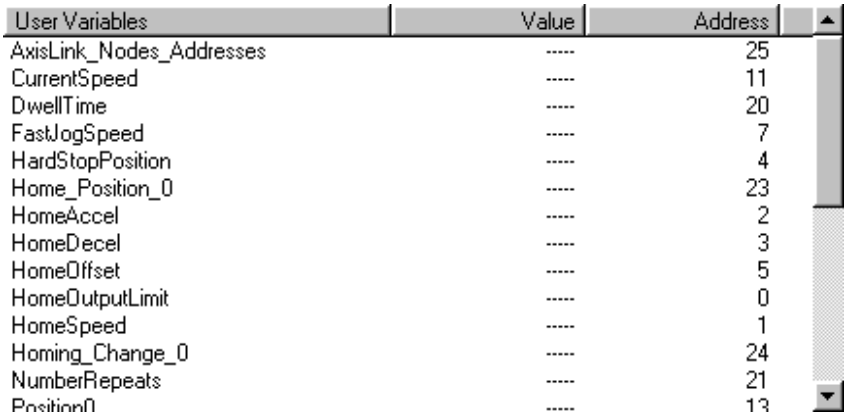
The Tag Explorer provides a hierarchical tree of tag categories, as shown in the example below:



You can navigate to any available category. The list of categories is dynamic and is dependent on the options available and selected in the Control Options dialog box. Selecting a category controls the detail view in the Tag Window.

Understanding the Tag Window

The Tag Window provides the framework within which individual tags are created and edited, as shown in the example below. The example shows the Axis 0 user variables that were defined.

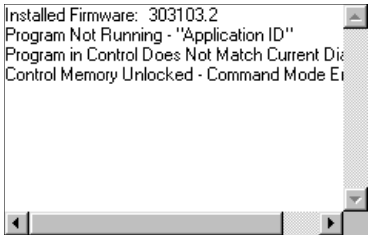


User Variables	Value	Address
AxisLink_Nodes_Addresses	-----	25
CurrentSpeed	-----	11
DwellTime	-----	20
FastJogSpeed	-----	7
HardStopPosition	-----	4
Home_Position_0	-----	23
HomeAccel	-----	2
HomeDecel	-----	3
HomeOffset	-----	5
HomeOutputLimit	-----	0
HomeSpeed	-----	1
Homing_Change_0	-----	24
NumberRepeats	-----	21
Position0	-----	13

When online the Tag Window also provides the framework within which individual tags are watched.

Understanding the Terminal Window

When online, the Terminal Window provides command/response links with the controller, as shown in the example below:



When you select normal mode, the viewed information is controller language independent. When you select terminal mode, the viewed information is shown in the controller's native language.

For information on using the online capability to communicate with your controller, see the *Going Online* chapter.

Script Editor Window

GML Commander provides the advantage of a graphical user interface to develop, edit and debug programs. Successful construction of complex programs is possible without having to resort to writing lines of code. After the diagram is completed, it is translated into a program called a script that is written in the native language of the motion controller. This application program is then downloaded to the motion controller.

While debugging a diagram, it is sometimes useful to view the translated program using the Script Editor Window to assure syntactical correctness. Programs may be printed for reference or saved for retrieval and download.

ATTENTION: We recommend that experience programmers use the script editor only when necessary. If you make changes to the script, they are not converted back to the diagram. An error could cause unexpected motion resulting in damage to the machine or personal injury.

For more information on using the Script Editor to view and fine-tune your program, see the *Working with Scripts* chapter.

Understanding Your Tools

Title Bar

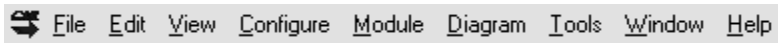
The GML Commander title bar shows the name of your diagram along with the size and close controls. Use the Title Bar to locate a diagram in the GML Commander application workspace. If you have not yet named or saved your diagram, the diagram is temporarily named New Diagram. For example, when you open a new diagram, you see the following:

See the *Working with Diagrams* chapter for information about naming your diagram.

After you save a diagram, the name changes, similar to the following:

Main Menu

The diagram menu located at the top of the window provides access to the primary GML Commander functions.



Refer to the following table for an overview of GML Commander's primary menus. See the On-line Help for more detail about GML Commander specific commands.

This menu:	Gives you access to:
File	Commands that apply to GML Commander (Properties and Exit) and diagrams (New Diagram, Open Diagram, Close, Save, Save As, Print, Print Preview and Print Setup).

This menu:	Gives you access to:
Edit	Commands for editing and making aesthetic changes (such as cutting, copying, pasting, aligning blocks) to your diagram as you develop it. You can also use the find function and access block details.
View	Commands that allow you to control the display of toolbars and status bar.
Configure	Commands that allow you to configure motion and communication features for your application.
Module	Commands for encapsulating blocks into modules and accessing module details.
Diagram	Commands for diagram testing, online functions and accessing diagram details.
Tools	Command that allows you to control the display of toolbars. Duplicates the same capability in View menu.
Window	All open GML Commander windows from which you can select the one you want to have focus.
Help	GML Commander's online help functions.

Main Toolbar

The Main Toolbar lets you quickly perform common tasks without the need to access the pull down menus. These tasks include:

- File menu commands i.e. New Diagram, Open Diagram, Save, Print
- Edit menu commands i.e. Cut, Copy, Paste, and Find.
- Display context sensitive help.
- Open the Document Module Window.
- Download the active iCODE script file to the controller.

By default the Main Toolbar is enabled when COmmander is invoked. You can disable or re-enable by selecting Toolbars from the View or Tools menu. You can click on any button on the toolbar to see its function.

Diagrams and Blocks

A diagram is a sequence flow of events and is composed of tags (definitions), blocks and connections. Tags describe the characteristics of data defined fields. Blocks represent actions and decisions. Connections link blocks in a specific order to show relationships and program flow. Blocks are copied to the diagram from one of the several block palettes.

Using the available tools, you create a diagram and print it for reference or save it for retrieval and later editing. Diagrams are translated to script to assure syntactical correctness.

Block Palettes

The color-coded building blocks that contain GML Commander functions are the graphical elements you use to create a diagram. These blocks are contained in logical groups called palettes. Palettes are selected for display and can be positioned anywhere on the screen in the same way as Windows toolbars.

For instructions on selecting and placing blocks in a diagram, refer to the *Working with Blocks* chapter. For detailed information on blocks, see the appropriate block chapter in this manual.

Color Code for Blocks

GML Commander uses color to group blocks by function.

Color	Function
Green	Initiate or change motion

Color	Function
Red	Stop motion
Yellow	Change setting or configuration
Brown	Affect program flow control
Pink	Provide I/O control
Violet	Provide operator interface functions
Light Green	Provide communications facilities
Light Yellow	Provide calculations, tables, and other similar functions
White	Provide miscellaneous functions that are not available in the categories above

A Module

A module is essentially a container for encapsulating one or more blocks. At the root level of each diagram is the main module consisting of a Start and End block with one or more functional blocks connected together. All modules in a diagram appear in the Diagram Explorer.

Testing Your Diagram

The Diagram Window also provides the framework within which application programs can be translated and tested. During program execution, the diagram shown in the Diagram Window can be traced, stepped and stopped.

When you are debugging a diagram's logic or performance, the visual progress of a trace program's execution is shown. Execution of specific functions and partial or total programs are monitored and supervised. Tools, for example, breakpoints, are provided for graphical debugging. A trace function highlights blocks as they are executed in the motion controller allowing you to follow the program execution. In addition, the ability to split the Diagram Window into two panes allows you to visually trace two paths, e.g., separate modules or separate tasks.

See the *Going Online* chapter for more information.

Naming a Block

When you first create a block, the description below it is generic. You can rename it using a descriptive name reflecting the block's function within this diagram. When you place the block, the description below it is highlighted. Click on the name area below the block and then type in the name you prefer. Clicking anywhere on the diagram releases the block.

Naming a Module

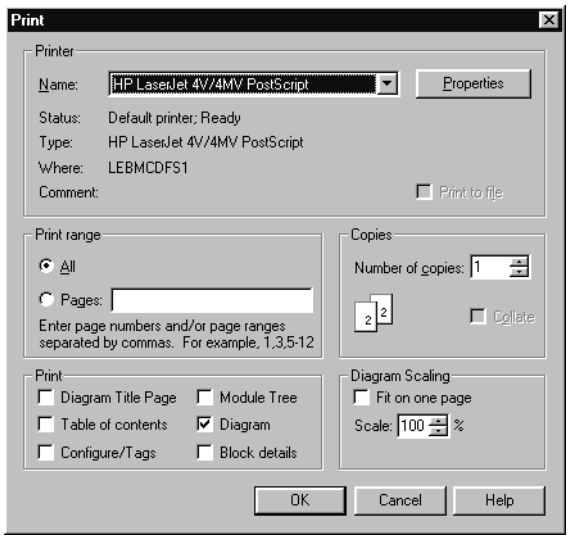
When you create a module, GML Commander assigns it the name *New Module x*. You can rename it using a descriptive name to reflect the module's function. To rename the module, highlight the name area by clicking on it with the mouse and then type in the preferred name. To release the module, click any blank area in the diagram.

Printing

Printing within GML Commander takes advantage of the common Windows printing tools. It uses your system defaults in determining which printer to use and its setup. We have added a few special features used in printing diagrams. This section explains how the special features in Commander work. See your Windows manual for specific questions on how Windows features work.

The Print box of the Print screen determines the degree of detail to print about the diagram.

.



Printing a Diagram

For each diagram, you can choose to print one or more of the following:

- **Diagram Title Page –** This option prints the Diagram Title page and the diagram synopsis. Use the **Diagram Title Page** command from the **Diagram Menu** to enter the heading and contents for the title page.
- **Table of contents** – Table of contents listing the page of each module's graphic depiction. The number of modules and blocks in the diagram and the number of times that each module is used also prints.
- **Configure/tags** – Description of the control options and values assigned to each axis. Also prints the address and name of each defined tag.
- **Module Tree –** Prints a textual representation of the module tree. The data is taken from the Diagram Explorer.
- **Diagrams** – Prints graphic depiction of the foundation diagram and each module as they appear in the Diagram Window.
- **Block details** – Prints a description of the parameter values for each block. The blocks for each module are printed on a separate page.

Diagram Scaling

When opting to print a diagram, you can choose how to handle the size of the diagram in relation to the page. The Diagram Scaling box of the Print Screen provides two options.

- **Fit on one page** – Select this to fit the picture of your diagram on a single page.
- **Scale** – Select this to scale the picture of your diagram to a percentage of its size as it appears on screen.

Select the choice that meets with your printing needs.

Chapter **2**

Configuring Control Options

Before beginning a new diagram, you must define basic control options by either using the configuration settings from an existing diagram or by selecting new settings. GML Commander uses this information to customize the menu options based on the controller you are using.

This chapter provides step-by-step procedures for defining required system configuration options that are common to all diagrams that you create. The main topics in this chapter include:

- Defining a diagram's configuration settings
- Setting the axes
- Setting the serial port interface
- Setting the Flex I/O

Defining a Diagram's Configuration Settings

To define a diagram's configuration, you can either copy a diagram that already has the configuration you need, or select from the GML Commander menu options and create the unique setting that you need.

Understanding the Precedence of Changing Setup Parameters

You can change setup parameters in a diagram block or from the Configure menu, depending on the extent of the change.

- If a parameter only pertains to the current diagram or a portion of the diagram, set a new or modified parameter in the diagram block.

- If you want to change the start-up value (also referred to as default or power-up value), enter a new parameter for the default control configuration using the Configure menu.

Using the Settings from an Existing Diagram

An easy way to configure your new diagram is to copy an existing diagram file that contains the proper configuration.

To use an existing diagram as the template for a new one:

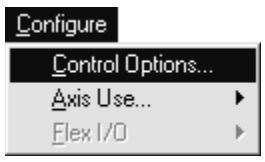
1. Select **File** from the menu bar. The File menu appears.
2. Select **Open Diagram**. The Open dialog box appears.
3. Select the diagram you want to copy for the new diagram. GML Commander diagrams have a **.**gml extension. The diagram opens.
4. Select **Save As**. The Save As dialog box appears.
5. In the *File name* field, type the file name for your new diagram.
6. Select **Save**.
7. You can now create or edit the new diagram. The original diagram file remains unaltered.

Selecting Settings for a New Diagram

You can also set up a configuration for a new diagram by selecting the parameters from the Configure Control Options menu.

To define the parameters for a completely new diagram:

1. Select **Configure** from the menu bar. The Configure menu appears.



Note: All options on this menu are part of the setup procedure. You can use these options later during testing and monitoring processes to make changes.

2. Select **Control Options**. The Configure Control Options dialog box appears.
3. To set the Control Options, follow the steps in the following section.

Setting Control Options

You configure the control options for the controller from the Configure Control Options dialog box.

Basic information required for all diagrams is presented on the General, Axes, and Interface tabs. Selections on the General tab can display additional tabs containing further options.

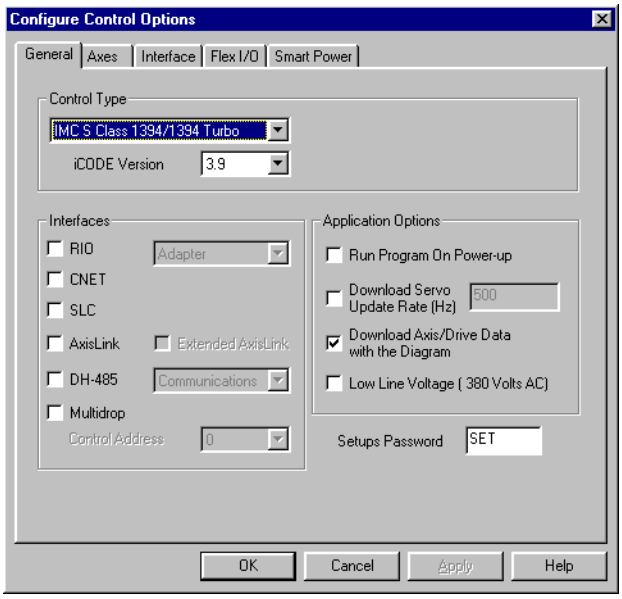
The tasks performed from this menu are:

- Setting the control type
- Setting the basic communications interfaces
- Setting the application options
- Defining the setups password

Setting General Control Options

To set control options for your controller:

1. Select **Configure** from the menu bar. The Configure menu appears.
2. Select **Control Options**. When the Configure Control Options dialog box displays, the basic tabs appear. The example below shows this dialog box before configuration settings are selected.



1. In the *Control Type* area of the General page, make entries in the following fields:

Field	**Description**	
Control Type	Select the type of your controller: • IMC S Class Basic/Integrated • IMC S Class Compact • IMC S Class 1394/1394 Turbo • MC S Class 1394L	
iCODE Version	Select the correct firmware version of your controller:	
	1394 GMC/GMC Turbo	3.0 or higher
	Compact	3.0 or higher
	Integrated/ Basic	2.3 or higher
	1394L	3.9 or higher

Field	Description
Expansion Card for AXIS2 and AXIS3	Select this option for Compact, Integrated, and Basic motion controllers if an expansion card is used. Note: This option is not available for 1394 GMC/GMC Turbo or 1394L.

2. In the *Interfaces* area, make entries in the following fields:

Field	Description
RIO	Selects an RIO interface for your controller. An RIO tab appears. For more information, see the *Configuring Your Remote I/O Interface* section later in this chapter.
CNET	Select this to enable the ControlNet communications option. CNET offers high speed deterministic communication of up to 99 nodes on the network. A CNET tab appears. **Note:** This option is only available if you have selected 1394 GMC Turbo. For more information, see the *Configuring Your CNET Interface* section later in this chapter.
SLC	Select SLC for high speed backplane communications between the motion controller and an SLC. An SLC tab appears. **Note:** This option is only available if you have selected 1394 GMC/GMC Turbo. For more information, see the Configuring Your *SLC Interface* section later in this chapter.

Field	Description
AxisLink	Selects AxisLink to enable communications for up to eight motion controllers on a network. An AxisLink tab appears. For more information, see the *Configuring the AxisLink Interface* section later in this manual.
DH-485	Select this to enable DH-485 operations for your controller. A DH-485 tab appears. Note: This option is not available for IMC S Class Basic/Integrated. For more information, see the *Configuring the DH-485 Interface* section later in this chapter.
Multidrop	Selects a Multidrop interface for your controller. No tab appears. For more information, see the *Configuring the Multidrop Interface section later in this* chapter.

3. In the *Applications Options* area, select the type of interfaces to be used:

Field	Description
Run Program On Power-up	The motion controller's program runs whenever power is applied. **Note:** Deselect this option when developing and debugging your diagram.
	**ATTENTION:** Do not use if RIO interface is selected and the PLC is running the program.

Field	Description
Download Servo Update Rate	1. Select this option to change the default Servo Update rate of 1000 Hertz to a new rate. Download Servo Update Rate affects the CPU utilization rate, as well as the frequency with which the CPU updates its reading of data, such as general system variable values, user variables, and I/O values. 2. Enter a different Servo Update rate (from 250-2000 Hz).

4. Place your cursor in the *Setups Password* field.

5. Select **SET**. This default password is highlighted.

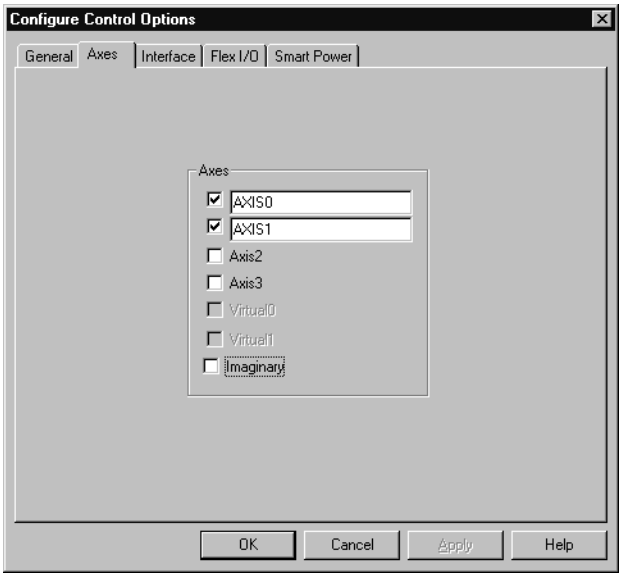
6. Type your password. **SET** is replaced with the new password.

Important: To save the new entries, select OK. This saves the information on all of the pages of the dialog box and closes it. If you select a different tab, your changes are not saved until you select OK on any page of the dialog box. If you select Cancel at any time, all changes that were made are lost.

Setting the Axes

To set axis options:

1. Select the **Axes** tab. The Axes page appears.



2. In the *Axes* area, select the axes you want to enable by clicking on the checkbox next to the axis name.

You can select these axes:	**Only when one of the following is selected:**
AXIS2 and AXIS3	• Expansion Card for AXIS2 and AXIS3 option on the General page. • 1394 GMC/GMC Turbo Control Type on the General page.
VIRTUAL0 and VIRTUAL1	AxisLink option on the General page.

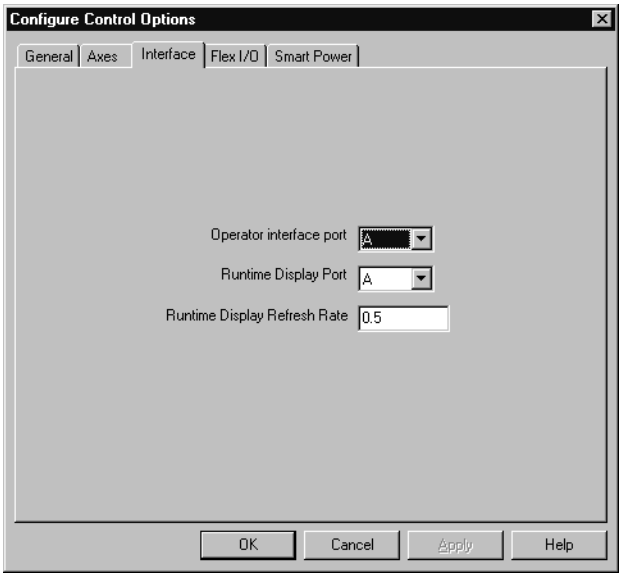
3. Rename the axes (optional).

Important: The axis name is appended to axis-specific system variables.

Setting the Serial Port Interface

To set the serial port interface:

1. Select the Interface tab. The Interface page appears. The example below shows the default settings.



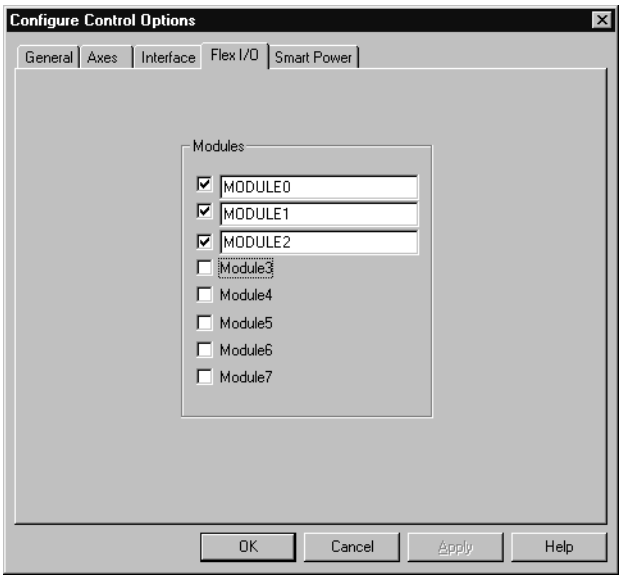
2. Make entries in the following fields:

Field	Description
Operator interface port	If the hardware configuration includes an optional operator interface terminal, select the controller serial port that is connected to that terminal.
Runtime Display Port	Select the serial port to use for the controller's runtime display.
Runtime Display Refresh Rate	Enter the frequency with which the controller refreshes the Runtime Display (in seconds). Because the human eye needs at least 0.2 seconds to recognize and respond to a display, refresh rates less than 0.2 seconds are not useful.

Setting the Flex I/O

To set Flex I/O:

1. Select the *Flex I/O* tab to display the *Flex I/O* page.



Note: Flex I/O capability is always present on Allen-Bradley 1394 and Compact motion controllers. Up to eight separate Flex I/O modules can be connected to a motion controller.

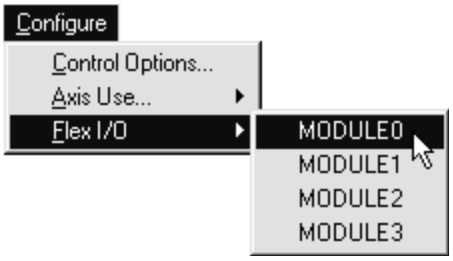
Note: In the *Modules* area, the first Flex I/O module connected is Module0, the second module connected is Module1, and so on. The names of selected modules can be changed for your application.

2. Select the modules you need.

3. Rename the Flex I/O modules (optional).

Defining Flex I/O

For every module selected in the *Modules* area of the Flex I/O page of the Configure Control Option dialog box, there is another selection added to the modules listed for the Flex I/O menu in the menu bar. An example is shown here. You must continue to define the Flex I/O module types and configuration options from this Flex I/O menu.



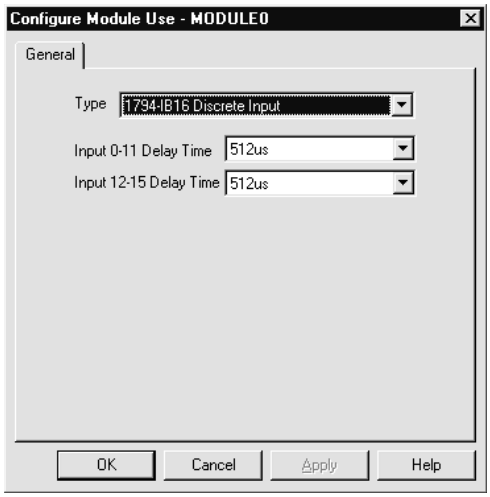
Important: The default module names are shown above. If you had renamed MODULE0 to Module Stop Test in the Flex I/O tab of the Configure Control Options dialog box, Module_Stop_Test would replace MODULE0 at the top of the Flex I/O module names.

To define Flex I/O:

1. Select **Configure,** from the menu bar. The Configure menu appears.
2. Select **Flex I/O**. A menu containing the names of the Flex I/O modules you've configured appears.

Important: Configure your Flex I/O starting with the first module (MODULE0) in the series and continuing until all selected modules are defined.

3. Select a module. A dialog box similar to the following appears.



4. Make entries in the following fields:

Field	Description
Type	The type of module you are assigning to this Flex I/O module. **Note:** the type of module selected here determines the fields for input and output that appear.
Entry Fields	The number of entry fields that display on this page are dependent on the kind of module selected in the Type field. Each field has a default value displayed. You may either accept the default or select an appropriate value from the pull-down list.

Note: A protected module is engineered to protect itself against damage from excess current (emanating from the system module) and excessive temperature. It shuts itself down if it encounters these conditions. However, such a module does not shut down and protect itself from the application of reverse current.

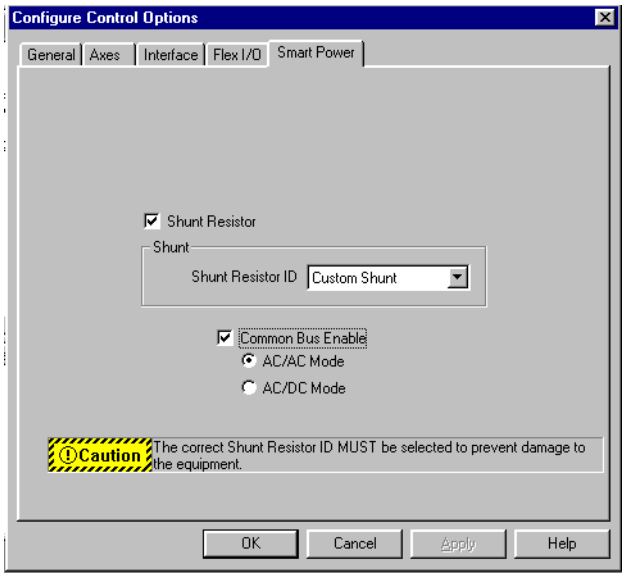
5.

If you:	Go to:
Do not have more modules to define	Step 6.
Have more Flex I/O modules to define	Step 1.

Setting the Smart Power Options

Select the Smart Power tab to make the necessary data parameter settings for a Smart Power 1394 system module. If your 1394 controller is not a Smart Power system module, make no selections on this page.

Note: This page is enabled only if Control Type IMC S-Class 1394/ 1394 Turbo is the selected Control Type and iCODE Version 3.7 (or higher) or 1394L is the selected Control Type and iCODE Version 3.9 (or higher) are selected in the General page of this dialog box.



To make the necessary System Smart Power data parameter settings (data parameters D132 through D136) and data bit (data bits B63 and B64) settings, make entries in the following fields:

Field:	Description:
Shunt Resistor	Select this if your Smart Power 1394 system module is equipped with a shunt resistor.Note:If your Smart Power 1394 system module is not equipped with a shunt resistor, do not make this selection.
Shunt Resistor ID	If Shunt Resistor is selected, above, select the catalog number of the Allen-Bradley shunt resistor you are using.
⚠	**ATTENTION:** If you do not select the correct Shunt Resistor ID, there is a risk of damage to the equipment.

Field:	Description:
Common Bus Enable	Select this if this Smart Power 1394 system module operates as a slave and is connected to another Smart Power 1394 system module (using DCLink hardware). In this configuration, one of the two system modules (the master) supplies DC power to another system module (the slave) over a common bus. Selecting this sets Data Bit #63 to 1. **Note:** Select Common Bus Enable only if this system is the Slave Module. Do not select it, if this module is the master system module.
Important: To use the Common Bus Enable feature, you must first select Shunt Resistor.	
AC/AC Mode	If Common Bus Enable is selected for this slave module, select AC/AC mode to configure this slave system module to receive 380/460-volt 3-phase direct AC power. Selecting this sets Data Bit #64 to 0.
AC/DC Mode	If Common Bus Enable is selected for this slave system module, select AC/DC mode to configure this slave system module to use only DC power. Selecting this sets Data Bit #64 to 1. In this configuration, this slave system module can not use 380/460-volt 3-phase direct AC power.

Note: If you are using a custom shunt resistor, select Custom Shunt and contact the manufacturer of the custom shunt resistor for the necessary data parameter settings (D132 to D136). Once these settings are obtained, use a series of Control Settings blocks to set these data parameters in your program.

Setting Optional Configurations

Use the information in this section to define additional configurations. This section provides step-by-step procedures to configure the following:

- An RIO interface
- A CNET interface

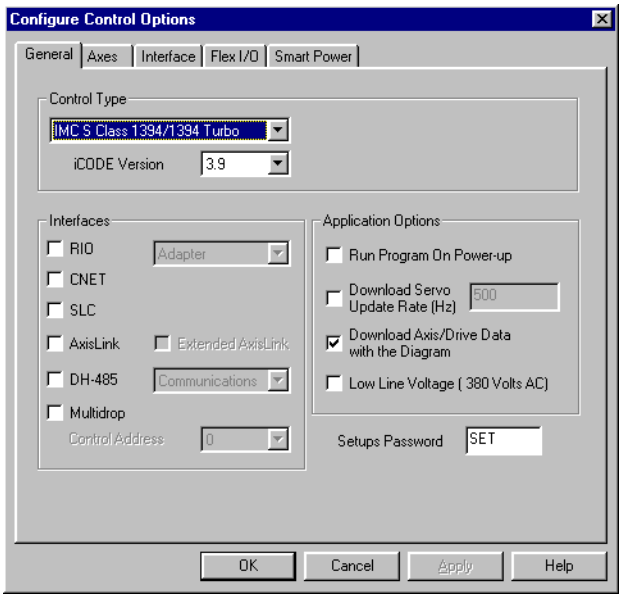
- An SLC interface
- An AxisLink interface
- A DH-485 interface
- A Multidrop interface

Like the settings you made for a controller's basic configuration, you set and modify these additional control options from the Configure Control Options dialog box.

You can create optional configurations from scratch or by copying from an existing diagram file with the same or similar configuration information that you want and modifying it to meet your needs.

To begin configuring optional interfaces for your controller:

1. Select **Configure,** from the menu bar. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears. The example below shows this dialog box before selecting configuration settings.



3.

To set up this communication interface:	Go to:
RIO	*Configuring Your Remote I/O Interface*.
CNET	*Configuring Your CNET Interface*.
SLC	*Configuring Your SLC Interface*.
AxisLink	*Configuring Your AxisLink Interface*.
DH-485 Link	*Configuring Your DH-485 Interface*.
Multidrop	*Configuring Your Multidrop Interface*.

Note: You can open another diagram to use as an example. To do this, open a second GML Commander application for the diagram that you want to use. Then open the same dialog box and size the window so that you can view both at the same time.

Configuring Your Remote I/O Interface

The Remote I/O (RIO) option provides RIO adapter functionality for all 1394 GMC, Compact, Integrated, and Basic motion controllers. This option allows the motion controller to communicate directly with an Allen-Bradley PLC or host computer via RIO.

Note: For IMC-S/20x-R and IMC/S21x-R models, the Remote I/O option also provides scanner functionality. For more RIO Scanner information, see the *Using the RIO Adapter Option* chapter of this manual.

When you use the RIO adapter you can control and monitor certain aspects of the motion controller from a PLC or SLC. RIO uses two transfer mechanisms:

To:	Use:
Exchange real-time data using discrete inputs and outputs.	Discrete transfers.
Transfer parameter values, recipes, status, and other non time-critical data for display purposes.	Block transfers.

To the PLC, the motion controller appears as ¼, ½, ¾, or a full remote rack on the RIO link. The following table shows the number of user-defined discrete I/O that is available depending on the rack size.

Rack size:	Inputs:	Outputs:	To make this part the starting group, select:
1st Quarter	4	4	0, 2, 4, or 6
Half	36	36	0, 2, or 4
3rd Quarter	68	68	0 or 2
Full	100	100	0

Configuring the RIO

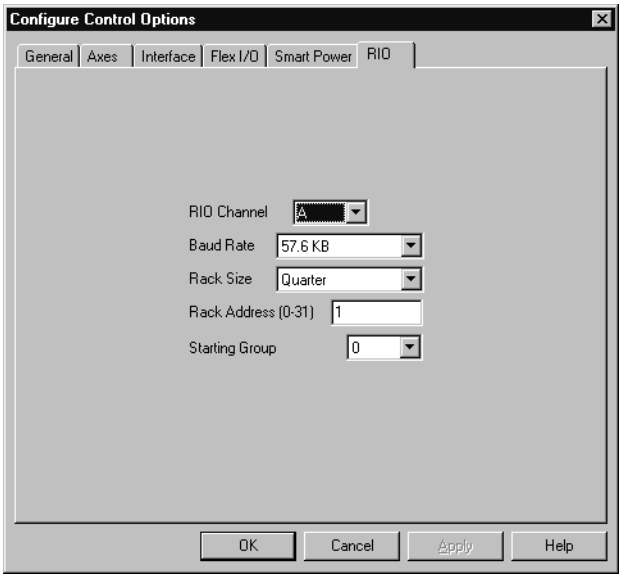
To configure the RIO interface:

1. At the Configure Control Options dialog box, select **RIO**.

2.

Select this RIO Mode:	To:
Adapter	Select to allow certain aspects of the motion controller's operation to be controlled and monitored from an Allen-Bradley PLC using a Remote I/O scanner. **Note:** This is the default selection for all models.
Scanner	Select to allow Remote I/O racks (PanelView or RediPanel) to be scanned by the motion controller. **Note:** This option is available as an alternative to the adapter mode for the IMC-S/20x-R and IMC/S21x-R models only.

3. Select the RIO tab. The RIO page displays.



4. Make entries in the following fields:

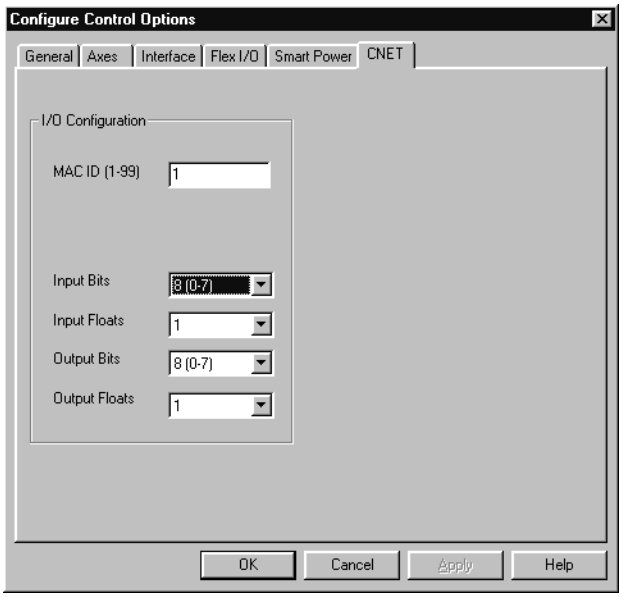
Field	Description
RIO Channel	Select the channel through which your controller communicates.
Baud Rate	Select the baud rate at which your controller communicates over the RIO link. **Note:** Every item on the RIO link must have the same baud rate.
Rack Size	Select the rack size for your controller that is based on the number of inputs and outputs you need. (See Rack Size for rack and I/O sizes.)
Rack Address	Type the rack address for this I/O connection, from 0 to 31 (octal), that represents the address of your controller. **Note:** Each rack is divided into four physical sizes. Refer to Rack Size above.
Starting Group	Select the area of the specific rack where you want the program to start. **Note:** Refer to Rack Size above.

5. Select **OK**. The selected options are stored as part of the current GML Commander diagram. The dialog box closes.

Configuring Your CNET Interface

To configure the CNET (ControlNet) interface:

1. Select CNET in the Configure Control Options dialog box. The CNET tab displays.
2. Select the CNET tab. The CNET page displays.



3. In the *I/O Configuration* area, make entries in the following fields:

Field	Description
MAC ID (1-99)	Enter the Media Access Control (MAC) ID, or address, of the controller. The value is an integer from 1 to 99.
Input Bits	Select the number of input bits you need from the pull down list. Available values are: • 8 (0-7) • 24 (0-23) • 40 (0-39)
Input Floats	Select the number of input floats you need from the pull down list. The available values are 0 to 15 (or 14 see note below table).

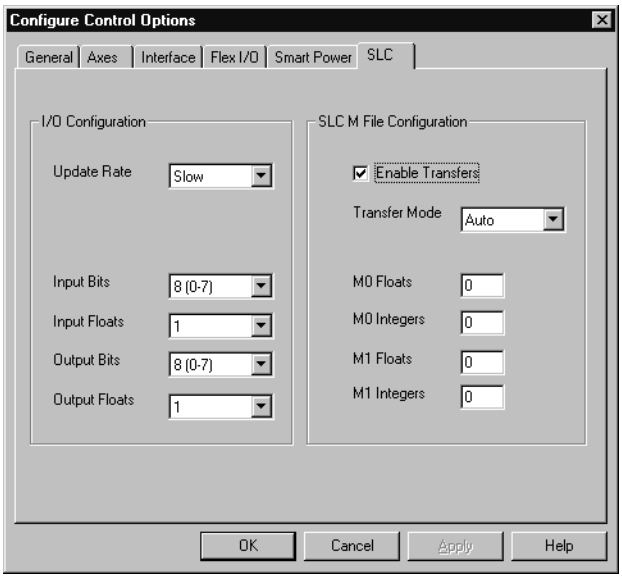
Field	Description
Output Bits	Select the number of input bits you need from the pull down list. Available values are: • 8 (0-7) • 24 (0-23) • 40 (0-39)
Output Floats	Select the number of input floats you need from the pull down list. The available values are 0 to 15 (or 14 see note below table).

Note: CNET I/O files have 32 words of I/O available. Each float consists of two 16 bit words. I/O floating point addressing starts at the word following the last discrete I/O. If you select 8 or 24 discrete I/O points, you can define up to 15 floats. If you configure all 40 I/O points, you can define up to 14 floats.

Configuring Your SLC Interface

To configure the SLC interface:

1. In the Configure Control Options dialog box, select **SLC**.
2. Select the SLC tab. The SLC page appears.



3. In the *I/O Configuration area*, make entries in the following fields:

Field	Description
Update Rate	I/O image file updates to and from the SLC are executed from the servo interrupt routine, and are independent of the SLC I/O scan. The higher the setting, the greater the impact on CPU utilization. Select one of the I/O update rates: Slow: Updates the complete I/O Image file every four servo loops. Medium: Updates the complete I/O Image file every two servo loops. Fast: Updates the complete I/O Image file every servo interrupt.

Field	Description
Input Bits	Select the number of input bits you need: • 8 (from 0 to 7) • 24 (from 0 to 23) • 40 (from 0 to 39)
Input Floats	Select the number of input floats from 0 to 15 (or 14. See the Note, below.) that you need.
Output Bits	Select the number of output bits you need: • 8 (from 0 to 7) • 24 (from 0 to 23) • 40 (from 0 to 39)
Output Floats	Select the number of output floats from 0 to 15 (or 14. See the Note, below.) that you need.

Note: In each SLC I/O file, 32 words of I/O are available . Each float consists of two 16 bit words. I/O floating point addressing starts at the word following the last discrete I/O. You can define up to 15 floats, if you select 8 or 24 discrete I/O points. You can define up to 14 floats, if you configure all 40 I/O points.

4. In the *SLC M File Configuration* area, make entries in the following fields:

Field	Description
Enable Transfers	Select this checkbox to allow information to be transferred between the 1394 and the SLC.
Transfer Mode	Select a transfer mode. **Note:** Refer to the *1394 GMC Turbo SLC Interface* chapter of the *GML Commander Reference Manual* (publication GMLC-5.2) for more information.

Field	Description
M0 Floats	Type the number of floats you need in the M0 file.
M0 Integers	Type the number of integers you need in the M0 file.
M1 Floats	Type the number of floats you need in the M1 file.
M1 Integers	Type the number of integers you need in the M1 file.

Note: The number of M0 and M1 words specified above should match the number of words set in the SPIO section of your SLC 500 program software.

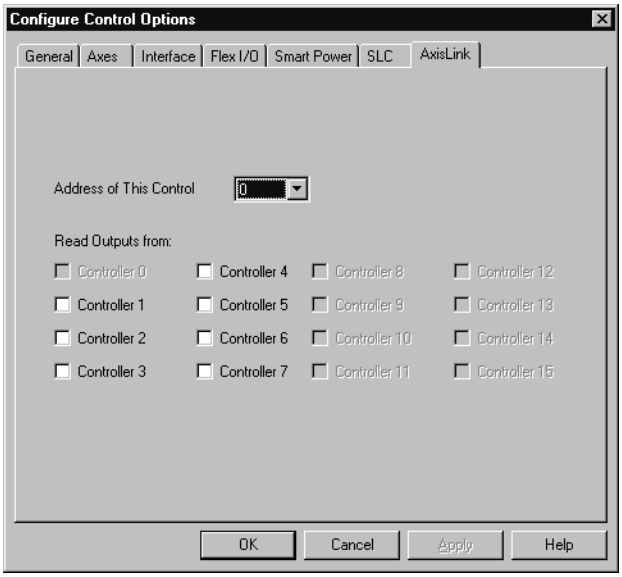
5. Select **OK**. The selected options are stored as part of the current GML Commander diagram. The dialog box closes.

Configuring the AxisLink Interface

AxisLink allows you to link multiple controllers together to exchange position and I/O information.

To configure your system for AxisLink:

1. Select **AxisLink** in the Configure Control Options dialog box.
2. Select the AxisLink tab. The AxisLink page appears.



3. Make an entry in the following field:

Field	Description
Address of This Control	Select the number representing the controller address for which you are developing this diagram.

4. Make entries in the following fields:

Field	Description
Controller *x*	Select the address(es) of the controller(s) from which you want this controller to receive outputs. **Note:** The address of the current controller (the one you assigned in the *Address of This Control* field) is grayed out.

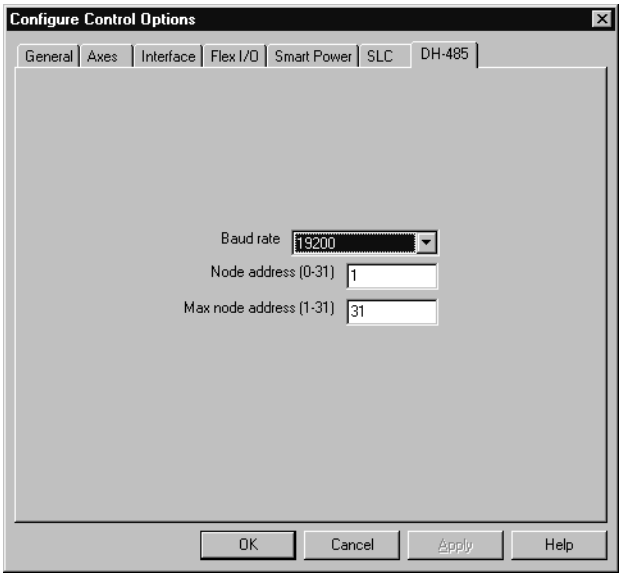
5. Select **OK**. The selected options are stored as part of the current GML Commander diagram. The dialog box closes.

Configuring the DH-485 Interface

The Data Highway (DH-485) interface enables communications capability between motion controllers, SLC controllers, and operator interfaces on a DH-485 communication network.

To configure the DH-485:

1. Select **DH-485** in the Configure Control Options dialog box.
2. Select **Communications** in the field to the right of the *DH-485* field.
3. Select the DH-485 tab. The DH-485 page appears.



4. Make entries in the following fields:

Field	Description
Baud Rate	Select the baud rate for your DH-485 network. **Important:** Every item on the link must have the same baud rate.
Node Address	Type the node address number (0 through 31) for this controller.
Max Node Address	Type the highest number of possible addresses for devices on this network.

5. Select **OK**. The selected options are stored as part of the current GML Commander diagram. The dialog box closes.

Configuring the Multidrop Interface

If you select Multidrop as an interface on the Configure Control Options dialog box, select a control address (0 through 7).

To configure the Multidrop parameters:

1. Select **Multidrop** in the Configure Control Options dialog box.
2. In the *Control Address* field, select a control address (0 through 7) to identify the control when it is communicating in RS-422 multidrop or daisy-chained mode. This control address must match the address selected with the rotary switch located on the front of the controller.
3. Select **OK**. The selected options are stored as part of the current GML Commander diagram. The dialog box closes.

Configuring Axis Use

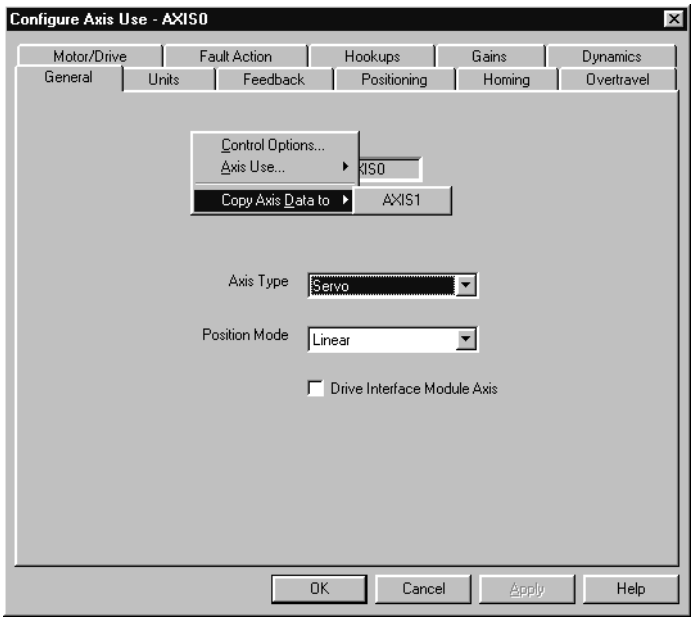
The following pages provide explanations for configuring axis use.

There are two ways to configure an axis. You can copy the configuration from an existing axis or you can define all the parameters for each enabled axis from the Axis Use menu.

Copying an Existing Axis Configuration

To copy an existing axis configuration to another axis follow these steps:

1. Select Configure from the Menu bar to access the Configure Menu.
2. Place the cursor over the Axis Use menu option. A pop up menu displays.
3. Select the axis with the configuration settings you want to copy. The Configure Axis Use dialog box opens for the selected axis.
4. Click the right mouse button. A pop-up menu appears.



5. Place the cursor over the **Copy Axis to,** this accesses the axis pop-up menu.
6. Select the axis to which you want to apply the axis configuration settings.
7. The settings are copied to the new axis.

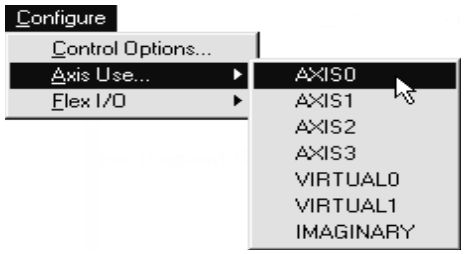
ATTENTION: This process overwrites the existing values for the designated axis. Even when hardware configurations are similar, certain values may need to be adjusted to ensure safe performance. Incorrect axis configuration values can result in unexpected motion, damage to the equipment, and personal injury.

To Configure an Axis With the Axis Use Screens

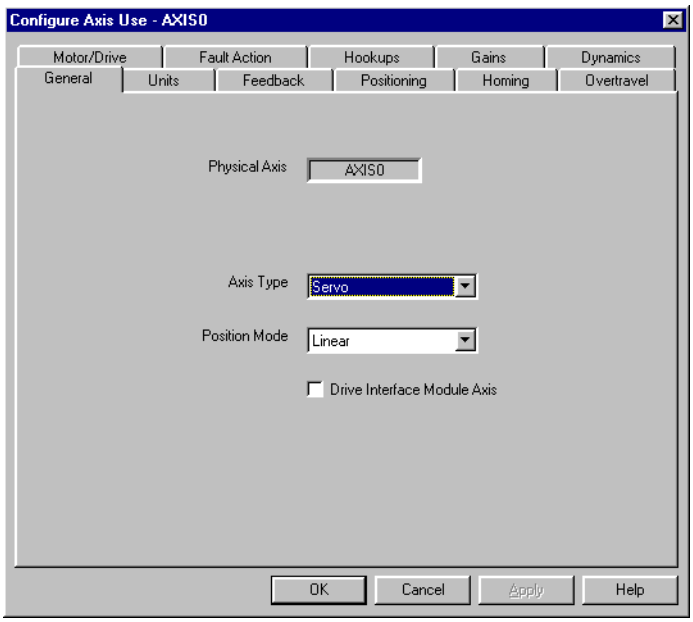
The number of tabs that appear for an individual Axis Use dialog box depends on whether the axis is a physical, imaginary, or virtual axis.

To configure a physical axis:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Axis Use**. The Axis Use menu appears.



3. Select an axis. The Configure Axis Use dialog box appears:



Note: Default names are shown in the examples above. If you had renamed AXIS0 to First on the Axes page of the Configure Control Options dialog box, First would replace AXIS0.

General

The General page of Configuring Your Axis is where you make the fundamental settings for an axis. The selections you make on this page determine how you can use this axis.

To configure the General page, make entries in the following fields:

Physical Axis

The default name of the physical axis (AXIS0, AXIS1, AXIS2, or AXIS3) is automatically displayed.

Axis Type

Select how the axis operates:

- Master Only, to use this axis only as a master axis.

Note: For Axis1 on IMC S Class 1394L controllers, Master Only is pre-set as the only permitted Axis Type.

- Servo, to use this axis as a servo axis, or as both a master axis and servo axis.

Position Mode

Select how to express forward motion along this axis:

- Linear: to continually increase axis position units during forward axial motion.
- Rotary: to increase axis position units during forward motion only to a pre-set number, then reset axis position to a starting value without backwards axis motion by subtracting a constant value (the Unwind Constant) from axis position.

The Rotary position mode feature, described above, is called Electronic Unwind. It provides infinite position range to the rotary axis, by unwinding the axis position (that is, returning axis position to a starting value) whenever the axis completes a cycle. The Unwind Reference and Unwind Constant values are set on the Feedback page.

Drive Interface Module Axis

You must select this for the first axis, and for every subsequent axis (in the AXIS0, AXIS1, AXIS2, AXIS3 sequence), enabled by a Drive Interface Module (DIM).

Note: The Drive Interface Module sends a ±10 Volt signal to up to four external drives and axes motors. It is used in place of one or more (up to four) axis modules.

To enable this selection, first select IMC S Class 1394/1394 Turbo or IMC S Class 1394L as the Control Type in the General Page of the Configure Control Options dialog box.

Defining Units

The Defining units page is where you select a particular position unit for a physical axis, and to format displays of physical axis Position, Velocity, Acceleration and Deceleration data. The displays you configure in this page include the runtime display and the motion controller's machine setup and servo setup display menus.

To configure the Units page, make entries in the following fields:

Position Units

Type an engineering unit (inches, meters, mm, degrees or revolutions) and not feedback counts to be used for measuring and programming all motion-related values.

Position Units can be different for each axis, and should meet the needs of your program. For example, you might select inches, meters or mm for a linear axis, and revs or degrees for a rotary axis.

Position Display Format

Type an axis position display format, using the # character as a placeholder and a decimal point (.) to display fractional values.

Note: Position Display format can contain up to 10 digits.

Velocity Display Format

Type an axis velocity display format, using the # character as a placeholder and a decimal point (.) to display fractional values.

Note: Velocity Display Format can contain up to 13 digits.

Acceleration and Deceleration Display Format

For a servo axis, type an acceleration and deceleration display format, using the # character as a placeholder and a decimal point (.) to display fractional values.

Note: Acceleration and Deceleration Display Format can contain up to 15 digits.

Defining Feedback

The Defining Feedback page is used to make settings that let GML Commander translate transducer or encoder counts to user-defined motion measurement units for a physical axis.

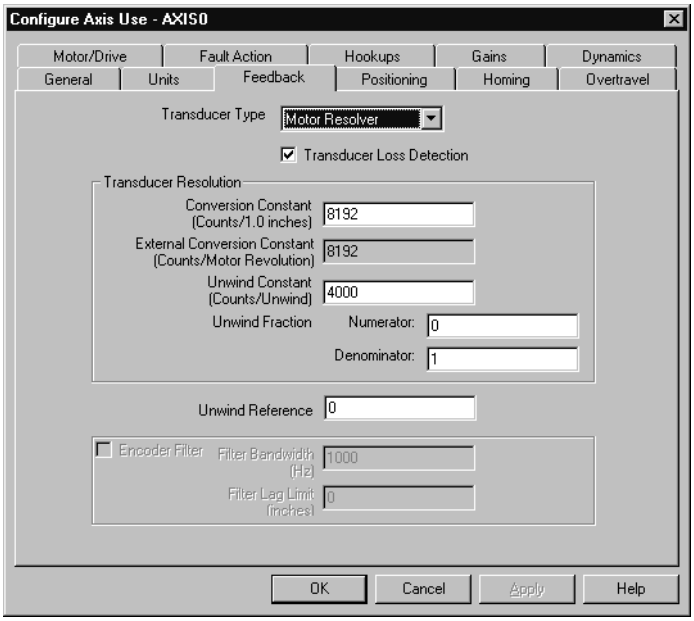
To configure the Feedback page, make entries in the following fields:

Transducer Type

Select the type of transducer or encoder that you are using.

Transducer Loss Detection

Click on the Transducer Loss Detection check box to activate it. When in the active state the controller detects any loss of transducer signal.



Transducer Resolution

Conversion Constant (K)

To allow axis position to be displayed and motion to be programmed in the position units specified in the Units Tab, a conversion constant must be entered for each axis. The conversion constant (K) lets you convert your position units into encoder (feedback) counts and vice versa. For physical and virtual axes, enter a value for K which is the number of encoder (feedback) counts per position unit using up to 10 digits.

The S-Class motion controllers use 4X encoder decoding (both edges of Channel A and B are counted). The count direction is determined from both the direction of the edge and the state of the opposite channel. Channel A leads Channel B for increasing count. This is the most commonly used decode mode with incremental encoders, since it provides the highest resolution.

For example, suppose this axis utilizes a 1000 line encoder in a motor coupled directly to a 5 pitch lead screw (5 turns per inch). With position units of Inches, the conversion constant is calculated as shown below:

K = 1000 lines/Rev * 4Counts/Line * 5Revs/inch = 20000 Counts/Inch

For the imaginary axis, the conversion constant is essentially arbitrary, but does affect the smoothness of gearing and position-lock cams which use the imaginary axis as their master. In general, a value between 4000 and 10,000 counts per position unit should provide adequate resolution.

Unwind

If the axis is configured as a rotary Axis at the General screen of Configure Axis Use, an Unwind value is requested. This is the value used to perform electronic unwind of rotary axes. Electronic unwind allows infinite position range for rotary axis by subtracting the unwind value from both the actual and the command position every time the axis makes a complete revolution.

The Unwind value is made up of the Unwind Constant (integer portion) and the Unwind Fraction.

Unwind Constant

If the axis is configured as a rotary Axis at the General screen of Configure Axis Use, a value for the Unwind constant is requested. This is the integer portion of the Unwind Value used to perform electronic unwind of rotary axes. The Unwind Constant plus the Unwind Fraction comprises the Unwind Value that determines the Unwind.

The unwind value is requested in units of encoder or feedback counts per axis revolution. Enter the appropriate value using up to 10 digits and press Enter.

For example, suppose this axis is configured as a rotary axis using position units of degrees and 10 feedback counts per degree. It is desired to unwind the axis position after every revolution. In this case the Unwind value would be 3600.

10 Counts/Degrees * 360 Degrees/Rev = 3600 Counts/Rev

Unwind Fraction

The Unwind Fraction lets you add a fractional count to the Unwind Constant to more accurately define the Unwind value. The ability to entire a fractional count for the unwind reduces the position error that can accumulate per cycle.

The Unwind Fraction is entered in two parts: the numerator and the denominator.

Numerator

The Unwind Fraction Numerator is the top value of the fraction being added to the Unwind Constant. Valid Numerator values range from:

- 0 to 999,999,999

The maximum permissible value is the Unwind Fraction Denominator value minus 1.

Denominator

The Unwind Fraction Denominator is the bottom portion of the Unwind Fraction. Valid Denominator values range from:

- 1 to 1,000,000,000

Unwind Reference Point

If the axis is configured as a rotary axis in the General section of Configure Axis Use, a value for the Unwind Reference point is requested. This value is used when an unwind is performed on a rotary axis. The Unwind Reference point is the position to which the axis rolls during an unwind. The axis rotation position is equal to the Unwind Reference Point plus the Unwind.

Rotation Point = Unwind Reference Point + Unwind

The total distance travelled before the axis unwinds remains constant and is not affected by the Unwind Reference Point.

For example, if the Unwind is 4 position units and the Unwind Reference Point is -1 position units, the axis rotates up to 3 position units and unwinds back to -1 position units.

The Unwind Reference value is entered in position and can be a positive or negative number up to 10 digits. The default value is zero (0).

Encoder Filter

The Encoder Filter filters encoder input signals from a Master Only axis (including a virtual axis) to other axes that are geared or pcammed to the master axis. The Encoder filter provides a means to minimize noise from externally controlled axes. Click on the Encoder Filter checkbox to activate the Encoder Filter. When activated, the two parameters, Filter Bandwidth and Filter Lag Limit, become active and must be set.

Filter Bandwidth

The Filter Bandwidth parameter sets the value to remove high frequency noise above a specified bandwidth. The value must be set between 1 and 1000 Hz. A general rule for determining the Filter Bandwidth is to keep a ratio for Filter Bandwidth to Servo Update at 1:10.

For example: If the controller has a servo update set at 500Hz the Filter Bandwidth should be set at 50Hz.

All frequencies higher than the input value are filtered-out when the encoder filter is ON .

Important: The ratio of Filter Bandwidth to Servo Update Rate (set in the General page of the Configure Control Options dialog box) should not exceed 1:10.

Filter Lag Limit

The Filter Lag Limit parameter sets the value for the maximum allowable filter induced lag which the controller cannot exceed. Enter a value representing the maximum permissible filter-induced lag in position units (as determined at the Units Tab) between the filtered output and actual position. Commander lets you enter a value between 0 and 1000 position units.Use the following formula to determine the filter induced steady state lag for a given constant velocity and filter bandwidth:

$$Lag = (velocity \times (BW \times servo\ period + 1)/BW) - (velocity \times servo\ period)$$

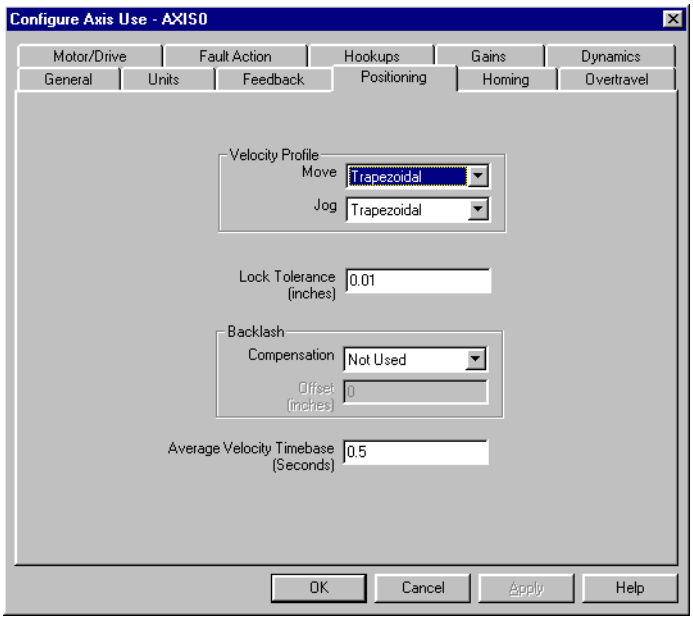
where:

- velocity = units per second
- servo period = seconds
- bandwidth (BW) = Hz.

Note: When filter-induced lag exceeds the input value, the controller turns the encoder filter OFF. When filter-induced lag falls below the input value, the controller turns the encoder filter back ON.

Defining Position

The Defining Position page is used for entering settings that determine a physical axis motion and positioning.



Velocity Profile

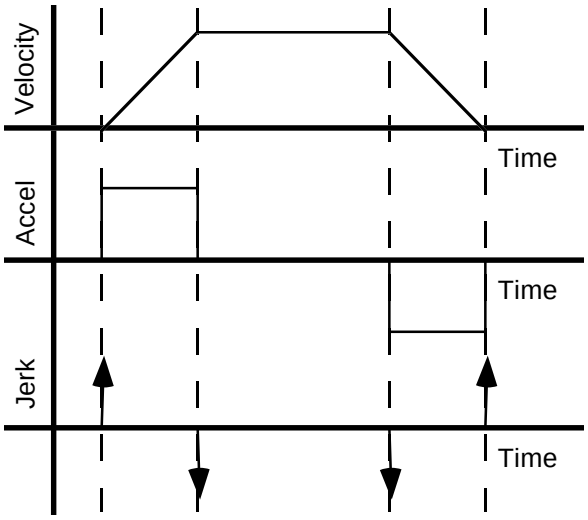
Velocity Profile is where you set the motion profiles for Move and Jog. The available motion profiles are:

- Trapezoidal - linear acceleration and deceleration
- S curve - controlled jerk
- Parabolic - conforms most closely to the torque/speed curve of most motors

Motion profiles are defined in the following manner,

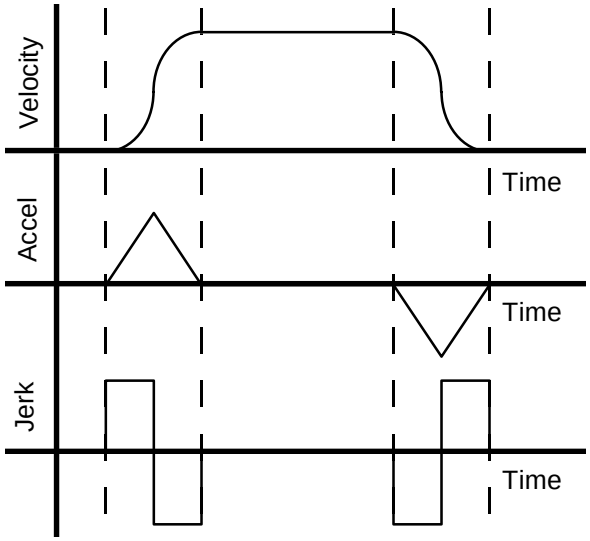
Trapezoidal

The trapezoidal velocity profile is the most commonly used profile because it provides the most flexibility in programming subsequent motion, and the fastest acceleration and deceleration times.



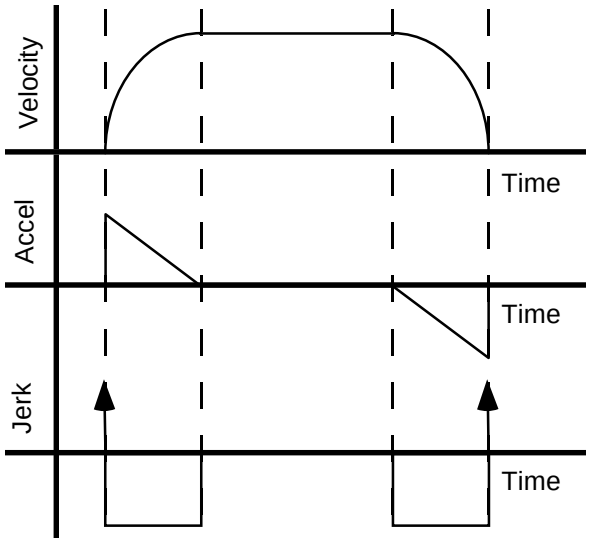
S Curve

You will most frequently use the S curve velocity profiles when the stress on the mechanical system and load needs to be minimized. However, the S curve velocity profile sacrifices acceleration and deceleration time compared to the trapezoidal. In addition, the speed (velocity) of S curve motion cannot be changed once the motion is started—except to zero—and the same acceleration and deceleration must be used. (Any deceleration value is ignored in subsequent Move or Jog Axis blocks.)



Parabolic

You will most frequently use a parabolic velocity profile to optimize the motor selection, because it conforms most closely to the torque/speed curve for most motors. However, the parabolic velocity profile sacrifices acceleration and deceleration time compared to the trapezoidal profile. In addition, the speed (velocity) of parabolic motion cannot be changed once the motion is started—except to zero—and the same acceleration and deceleration rate must be used. (Any deceleration value is ignored in subsequent Move or Jog Axis blocks.)



To configure the Positioning page, make entries in the following fields:

Move

Select the default motion profile for servo axis motion commanded by a Move Axis block:

- Trapezoidal
- S-Curve

- Parabolic

Jog

Select the default motion profile for servo axis motion commanded by a Jog Axis block:

- Trapezoidal
- S-Curve
- Parabolic

Lock Tolerance

Type a value, in position units, representing the maximum servo axis position error the controller will tolerate when indicating axis locked status.

Note: The controller interprets this value as a ± measurement around the axis position.

Backlash

Backlash is the relative movement between interacting mechanical parts due to looseness. Commander provides the means to set a compensation type and offset value to counter any looseness in your servo system.

Compensation

If your servo system contains slop (i.e. inaccuracy between commanded and actual position due to loose mechanical connections), you can compensate for this slop by selecting a backlash compensation type. the available types are:

- Positive Approach
- Negative Approach
- Load Reversal

Offset

If you selected backlash compensation, type a value equal to the distance, in servo axis position units:

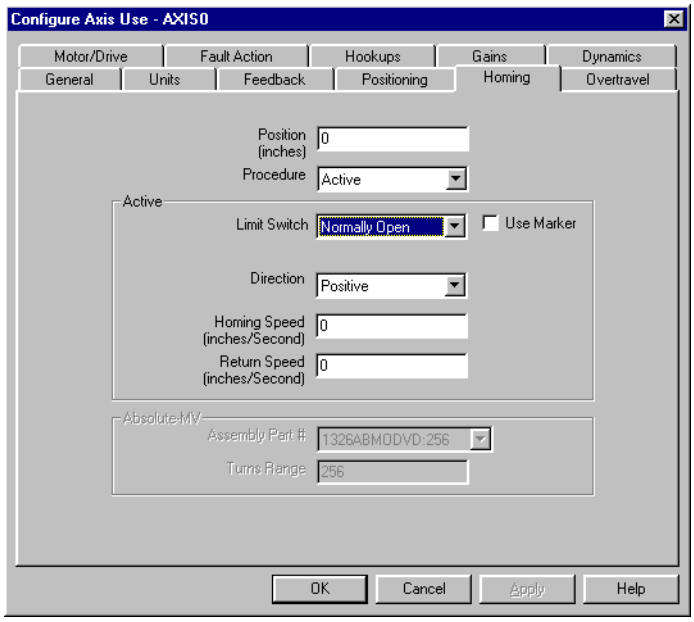
- the axis should overshoot when approaching the destination position from the opposite direction (for Positive Approach or Negative Approach), or
- to add/subtract from the axis' absolute position whenever the axis changes direction (Load Reversal).

Average Velocity Timebase

Type the time, in seconds, over which the controller will take 10 sample velocity readings as a basis for computing average velocity.

Defining Homing

The Defining Homing page is where you configure default homing settings for a physical axis. When a Home Axis block executes, with Configured selected, the settings made in this page control homing.



Position

Position refers to the reference position for all positioning movements (home position) of the axis. Type a value, in position units, that is the home position for this axis.

Procedure

The Homing procedure defines the method used to determine the position of the axis at startup. The procedures available are dependent upon the type of transducer selected in the Feedback portion of the Configure Axis Use. The homing procedures include:

- Active
- Passive
- Absolute

- Absolute Serial: only for AEC-216 and AEC Transducer Types.
- Absolute MV (Motor Vernier): only for REC Transducer Type.

Important: The selection of a Transducer Type, in the Feedback page, determines which homing procedures appear in this list.

Active Homing

Active homing is the most common homing procedure for physical servo axes. When ACTIVE is selected, the desired homing sequence is selected by specifying whether or not a home limit switch and/or encoder marker is used for this axis. Active homing sequences always use trapezoidal velocity profile. The available active homing sequences are described below.

Homing Without a Limit Switch or Marker

This is the simplest active homing sequence. When this sequence is performed the home position is immediately assigned to the current axis position. This homing sequence produces no axis motion.

Homing to a Limit Switch

This active homing sequence is useful for multi-turn rotary and linear applications where there are multiple encoder markers over full axis travel or when an encoder marker is not available.

When this sequence is performed, the axis moves in the specified home direction at the specified homing velocity until the home limit switch is detected. The axis decelerates to a stop and then moves in the opposite direction at the specified return velocity until the home limit switch is cleared, and the axis decelerates to a stop. The axis then moves back to the home position at the return velocity. The motions for this active homing sequence are shown below.

Neglecting the mechanical uncertainty of the home limit switch, the accuracy of this homing sequence depends on the time uncertainty in detecting the home limit switch transitions. The position uncertainty of the home limit switch (8 milliseconds) and the specified return velocity.

For example, if a return velocity of 0.1 inches per second (6 IPM) is specified, the uncertainty of the home position is calculated as shown below;

0.1 Inch/Second * 0.008 Seconds = 0.008 Inch

Homing to an Encoder Marker

This active homing sequence is useful for single turn rotary and linear encoder applications since these have only one encoder marker for full axis travel. When this sequence is performed, the axis moves in the specified home direction at the specified homing velocity until the marker is detected. The home position is assigned to the axis position corresponding to the marker location, and the axis decelerates to a stop. The axis then moves back to the home position at the specified return velocity. The axis motions for this homing sequence are shown below.

Homing to a limit Switch and Marker

This is the most precise active homing sequence available. When this sequence is performed, the axis moves in the specified homing velocity until the home limit switch is detected. The axis then decelerates to a stop and moves in the opposite direction at the specified return velocity until the home limit switch is cleared. After clearing the home limit switch, the axis continues in the same direction at the return velocity until the first encoder marker is detected. The home position is assigned to the axis position at the moment the marker is detected, and the axis then decelerates to a stop. The axis then moves back to the home position at the return velocity. Axis motions for this homing sequence are shown below.

When Active Homing is chosen and either Limit Switch or Marker are selected the following fields become active.

Direction

The Direction field sets whether Positive or Negative is used for the default homing direction of the axis.

Homing Speed

Type a default speed, in axis position units per second, at which the axis will move toward a limit switch or encoder marker in an Active homing procedure.

Return Speed

Type a default speed, in axis position units per second, at which the axis will return from a limit switch or encoder marker in an Active homing procedure.

Absolute Homing

If you are using an absolute feedback device (resolver, magnetorestrictive transducer or absolute encoder) that is interfaced to the controller through one of the available absolute transducer converters, select Absolute as the homing Procedure. When you select Absolute, the current absolute position of the feedback transducer is requested and used to initialize axis position at power-up and during a homing procedure.

During a homing procedure, the Absolute selection:

1. disables feedback,
2. activates the absolute position strobe output to read the current absolute position of the axis from the absolute transducer converter,
3. sets the actual position of the axis to the current absolute position of the feedback device plus the home position, then
4. re-enables feedback.

Note: There is an approximately two-second delay with absolute homing to allow the absolute position to be read from the absolute transducer converter.

Note: With absolute homing, the home position parameter acts as an offset to the absolute position read from the absolute transducer converter.

Absolute_MV

Absolute_MV is an absolute homing procedure. You can only use this procedure with a transducer type of REC. When the Absolute MV homing procedure is selected, the following fields become active.

Assembly Part #

For an Absolute MV homing procedure, select

- the default master resolver part number from the pulldown list, or
- Custom, if you are using a different master resolver.

Turns Range

If Custom is selected, the Turns Range field becomes active and an integer with a value between 1 and 1,000,000,000 must be entered. See the manufacturer's specifications for the correct value for the Turns Range.

Absolute Serial

Absolute Serial is a specific homing procedure. You can only select Absolute Serial when you are using a serial output absolute encoder and AEC is selected as the transducer type. When Absolute Serial is selected as the Homing Procedure, the Turns Range field becomes active. You must enter the turn range for the type of encoder/transducer that is used. This is must be an integer between 1 and 1,000,000,000. See the manufacturer's specifications for this value.

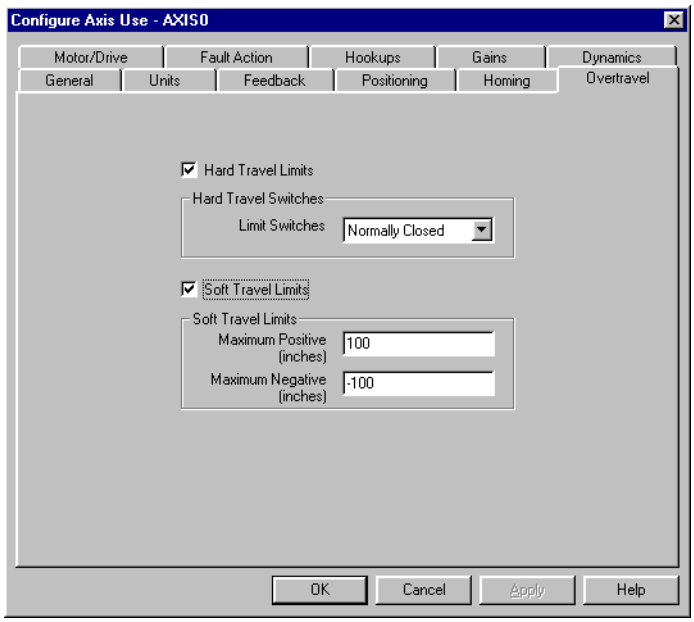
Passive Homing

Passive homing redefines the current absolute position of the axis upon the next occurrence of the encoder marker. You will most commonly use passive homing to calibrate Master Only axes to their markers (although you can use passive homing for Servo axes, too). Passive homing is identical to active homing to an encoder marker, except that no motion is commanded instead, the controller merely waits for the marker to occur.

Important: When a passive homing procedure for a rotary axis begins, the working value of axis mode is set to Linear during the procedure. When the passive homing procedure finishes, axis mode is reset to Rotary. However, if the passive homing procedure does not finish, axis mode remains set to Linear. In this case, be sure to reset the working value of axis mode to Rotary using a Control Settings block, with Adjust Type, Working Value, and the Rotary Axis Data Bit selected.

Defining Overtravel

The Defining Overtravel page is used to configure both overtravel limit switch checking and software travel checking to prevent a physical axis from traveling outside its pre-defined limits and possibly causing damage.



ATTENTION: In addition to setting hard and soft overtravel limits, hardwire the overtravel limit switches and/or the CPU watchdog contacts into the drives so that drive power is interrupted if a fault occurs. Otherwise, unpredictable motion can occur.

To configure the Overtravel page, make entries in the following fields:

Hard Travel Limits

Select this to use travel limit switches.

Note: If you select this, the action selected in the Fault Action page for Hard Overtravel will occur when axis travel touches a travel limit switch, thereby causing a Hardware_overtravel_fault .

Limit Switches

If you are using overtravel limit switches, select whether the limit switch contacts are:

- Normally Open
- Normally Closed

Note: Your selection applies to both limit switches for a physical axis.

Soft Travel Limits

Select this to use software travel limits.

Note: If you select this, the action selected in the Fault Action page for Soft Overtravel will occur when a software travel limit is reached, thereby causing a Software_overtravel_fault .

Maximum Positive

If you selected Soft Travel Limits, type the positive direction software travel limit.

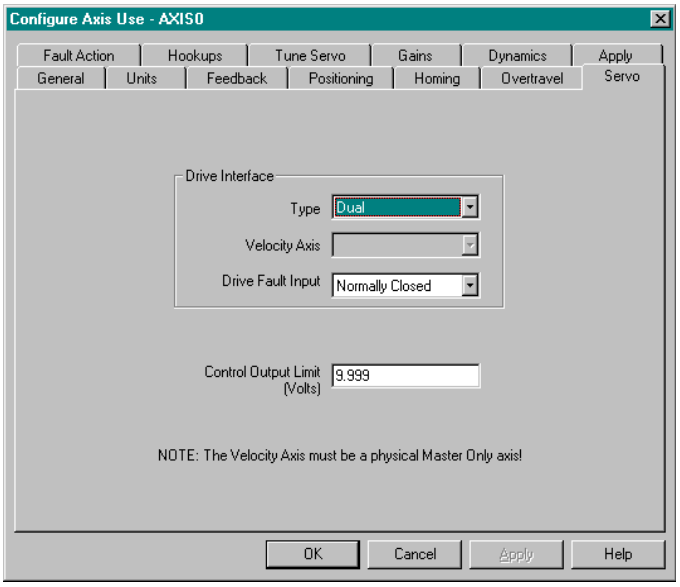
Maximum Negative

If you selected Soft Travel Limits, type the negative direction software travel limit.

Important: The Maximum Negative value must be less than the Maximum Positive value.

Defining Servo

The Defining Servo page is where you configure a physical axis to interface with a servo drive (or amplifier) operating in either velocity (tach) or torque (current) mode.



This page applies to axes controlled by:

- IMC S Class Compact controllers,
- IMC S Class Basic/Integrated controllers, and

- IMC S Class 1394/1394 Turbo controllers connected to a Drive Interface Module (DIM).
- IMC S Class 1394L controllers connected to a Drive Interface Module (DIM).

To configure the Servo page, make entries in the following fields:

Type

Select the servo drive type that matches the type of servo system you are using:

- Torque: the drive merely activates the motor, while the controller corrects for both position error (by closing the position loop) and velocity error (by closing the velocity loop).
- Velocity: the controller corrects for position error (by closing the position loop) and the drive corrects for velocity error (by closing the velocity loop).
- Dual : the controller corrects for both position error and velocity error using two separate feedback devices (encoders).

Velocity Axis

If you selected Dual, select a master only axis the controller will use to correct for velocity error by closing the velocity loop.

Drive Fault Input

Select how the motion controller uses the dedicated drive fault input for this physical axis:

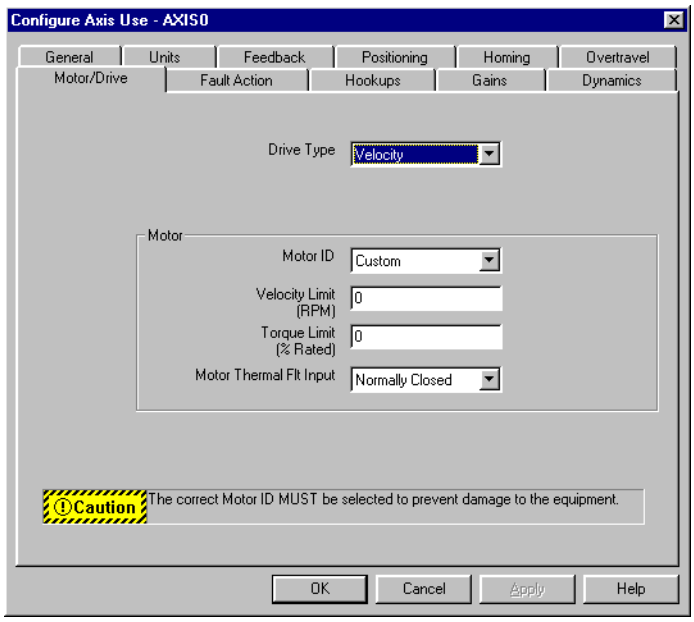
- Not Used: if the input is not used.
- Normally Open: if the drive activates the input when a Drive_fault occurs.
- Normally Closed: if the drive deactivates the input when a Drive_fault occurs.

Control Output Limit

Type the maximum number of volts that is output to the drive, from: 0 to 10.

Defining Motor/Drive

The Defining Motor/Drive page configures a physical axis to interface with both a motor and a servo drive (or amplifier) operating in either velocity (tach) or torque (current) mode. This page applies only to IMC S Class 1394/1394 Turbo and IMC S Class 1394L controllers.



To configure the Motor/Drive page, make entries in the following fields:

Drive Type

Select the drive type that matches the type of system you are using:

- Torque: the drive merely activates the motor, while the controller corrects for both position error (by closing the position loop) and velocity error (by closing the velocity loop).
- Velocity: the controller corrects for position error (by closing the position loop) and the drive corrects for velocity error (by closing the velocity loop).

Motor ID

Select from the scrolling list:

- the motor ID number for your motor, or
- Custom, if your motor's ID is not listed.

Optimizing Velocity & Torque Limit Settings

GML Commander uses the motion controller's limiting algorithms to generate Velocity Limit and Torque Limit settings. Velocity and torque limits are inversely related: the larger the torque limit, the smaller the velocity limit, and vice versa (until an absolute velocity limit for your motor is reached). Maximum Velocity for your system can be limited by:

- The power rating of your drive (amplifier)
- The capacity of your motor
- The Torque Limit Setting (as noted above).

To optimize Velocity and Torque Limit settings, follow these steps:

For listed motors:

1. Be sure you are operating online (select Online Connection in the Diagram menu or Online Toolbar if you are not online).
2. In the Motor/Drive page of the Configure Axis Use dialog box, select your Motor ID. GML Commander automatically inputs compatible Velocity Limit and Torque Limit values.
3. In the Torque Limit input box, type a value less than 300%.
4. In the Velocity Limit input box, type a large value (e.g. 10000 RPM),

5. Place the pointer in the Torque Limit data input box, and click the left mouse button. This brings focus to the Torque Limit field. GML Commander automatically generates a new Velocity Limit for the lowered Torque Limit value.

6. Repeat steps 3 through 5 until you reach the desired Velocity Limit (or absolute limit) for your motor.

For Custom motors:

1. At the beginning of your diagram, create a module that contains a Control Settings block, with Adjust Type selected, for each of the following data parameters:

D91 Motor Inertia 1394

D92 Motor Rated Speed 1394

D93 Motor Rated Current 1394

D94 Motor ID Start RPM 1394

D95 Motor ID End RPM 1394

D96 Motor ID Slope 1394

D97 Motor Resolver Offset 1394

D104 Motor RPM 100% Rated 1394

D105 Motor RPM 300% Rated 1394

D111 Motor Poles 1394

D112 Resolver Poles 1394

Refer to your motor's documentation for parameter values.

2. Follow steps, 1 through 6, for listed motors, above.

Note: If you select a motor ID from the list, GML Commander will make the Velocity Limit and Torque Limit settings for you.

If you select Custom, you must:

- Type the Velocity Limit and Torque Limit settings that apply to your motor, and
- Contact the manufacturer of the custom motor for the 11 motor-related data parameter settings (D91 to D97, D104 to D105, and D111 to D112).

Once these settings are obtained, use a series of Control Settings blocks to set these data parameters in your program.

ATTENTION: If you do not select the correct motor ID there is a risk of damage to the equipment.

Velocity Limit (RPM)

For Velocity Drive Type:

- If you selected a motor ID from the list, accept the default setting, or type a velocity limit in RPM.
- If you selected Custom from the Motor ID list, you must type a velocity limit in RPM.

Note: Velocity Limit and Torque Limit values are inversely related. If you input these values, refer to your motor documentation to insure these settings are compatible.

Torque Limit(% Rated)

If you selected a motor ID from the list, accept the default setting, or type a limit as a % of rated torque (0 to 300%).

If you selected Custom, type a limit as a % of rated torque (0 to 300%).

Motor Thermal Fault Input

Select whether you are using the dedicated motor thermal fault input that detects if the motor is overheating:

Not Used: if this input is not used.

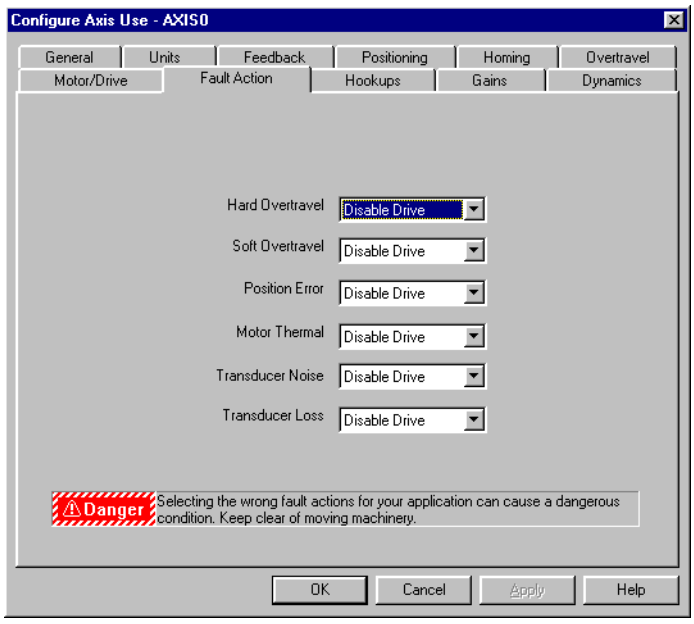
Normally Closed: if this input is used. (Because the input's contact is normally closed, the input is activated at system power-up, and deactivated when a Motor_Thermal_Fault_1394 occurs.)

ATTENTION: If you do not use the motor thermal fault as instructed by your motor documentation, damage to equipment can result.

Defining Fault Action

The Defining Fault Action page selects the action the controller takes in response to each of seven different faults that can occur when you run an application program. For each fault type (except for Transducer Loss) there are three selections

.



Disable Drive:

- Feedback is disabled.
- The controller cannot command motion.
- Motion on the axis depends upon your hardware configuration. Some hardware configurations may require brakes.

Stop Motion:

- Feedback remains enabled.
- The controller continues to be able to command motion.
- Motion stops at 100% of the Maximum Deceleration rate (set in the Dynamics page of the Configure Axis dialog box).

Status Only:

- Feedback remains enabled.

- Motion continues.
- The controller indicates a fault status.

ATTENTION: Selecting the wrong fault action for your application can cause a dangerous condition. Keep clear of moving machinery.

To configure the Fault Action page, make one of the above three entries in the following fields:

Hard Overtravel

Select the action that occurs when servo axis travel touches a travel limit switch, thereby causing a Hardware_overtravel_fault.

Note: Enable the use of travel limit switches in the Overtravel page of this dialog box.

Soft Overtravel

Select the action that occurs when servo axis travel reaches a software travel limit, thereby causing a Software_overtravel_fault.

Note: Enable the use of software travel limits in the Overtravel page of this dialog box.

Position Error

Select the action that occurs in response to a servo axis Position_error_fault .

Note: A position error fault occurs when the difference between Command Position and Actual Position exceeds the Error Tolerance set in the Dynamics page of this dialog box.

Drive Fault

Select the action that is to occur in response to a servo Drive_fault, if you enabled the Drive Fault Input in the Servo page of this dialog box.

Important: For IMC S Class Basic/Integrated and Compact controllers only.

Motor Thermal

Select the action that occur in response to a Motor_Thermal_Fault , if you enabled the Motor Thermal Fault Input in the Motor/Drive page of this dialog box.

Important: For IMC S Class 1394/1394 Turbo controllers only.

Transducer Noise

Select the action that is to occur if the transducer (encoder) interface receives erroneous signals due to electronic noise, causing an Encoder_noise_fault.

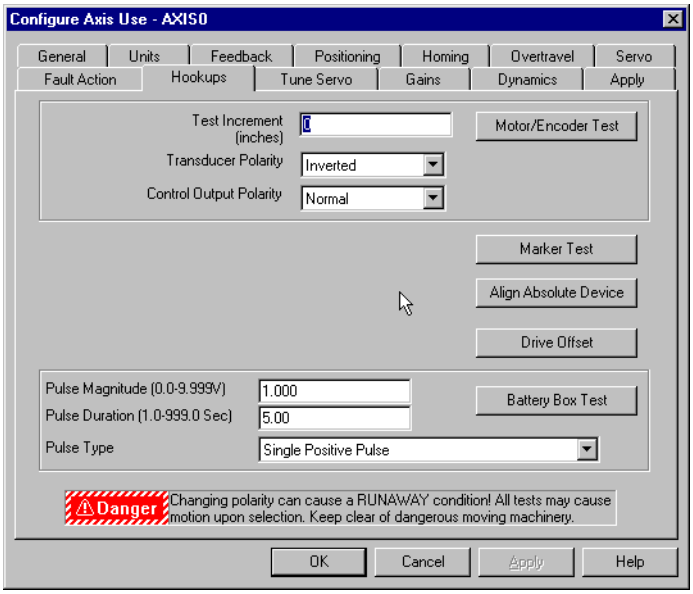
Transducer Loss

Select the action that is to occur if communications between the controller and transducer (encoder) is lost, thereby causing an Encoder_loss_fault, if you enabled Transducer Loss Detection in the Feedback page of this dialog box.

Note: Stop Motion is not a selection, because communications between the controller and transducer is lost and the controller cannot command motion to stop.

Verifying Hookups

The diagnostic routines on the Verifying Hookups page determine and verify the correct hookup of feedback transducers, servo drives (amplifiers) and limit switches.



Before running any of the diagnostic routines in this dialog box, be sure that:

- all external components are properly connected to your system,
- the GML Commander is operating in ONLINE mode, and
- the program, including all configured Axis and Drive data, in GML Commander is the same as the program in the controller.

ATTENTION: Failure to download can cause uncontrolled motion when performing controller diagnostic routines.

To configure the Hookups page, make entries in the following fields:

Test Increment

Type a distance value, in servo axis position units.

Note: In most cases, a distance value that is the equivalent of 1/4 of a motor revolution works well.

Motor/Encoder Test

Use the motor/encoder test for servo axes to check the proper electrical connection of the servo drive and encoder (or other feedback device), and to establish the correct rotational direction of the servo drive and encoder. Establishing these motor and encoder polarities insures the axis can not run away when the feedback loop is closed.

Performing the Motor/Encoder Test

Be sure all external components are connected and the program, including all configured Axis and Drive data, is downloaded.

ATTENTION: Failure to download can cause uncontrolled motion when performing controller diagnostic routines.

Then, do the following:

1. In the Hookups page of the Configure Axis Use dialog box, enter a distance in position units (between 0.001 and 999.999) as the Test Increment. (A value representing 1/4 of a motor revolution usually works best.)
2. Select the Motor/Encoder Test button. Watch the drive and positional readout.
3. Follow the directions in the message box prompts.
4. When the message box reads Motor Test OK and asks the question Observed Motion Direction Positive?, select:

- Yes if the axis moved in the intended direction.
- No if the axis moved in the negative direction.

5. Based upon your response, GML Commander sets the values in both the Transducer Polarity and Control Output Polarity fields.

Transducer Polarity

This field is set by successfully performing the Motor/Encoder test.

ATTENTION: Changing the Transducer Polarity after the motor/encoder test has been successfully completed can cause a RUNAWAY condition.

Control Output Polarity

This field is set by successfully performing the Motor/Encoder test.

ATTENTION: Changing the Control Output Polarity after the motor/encoder test has been successfully completed can cause a RUNAWAY condition.

Marker Test

If you have configured the controller to use the encoder marker for homing (in the Homing page of this dialog box), use the marker test to verify that the controller sees the marker.

How to perform the Marker test

Be sure all external components are connected and the program, including all configured Axis and Drive data, is downloaded.

Then, do the following:

1. In the Hookups page of the Configure Axis Use dialog box, select the Marker Test button.
2. When prompted by GML Commander, manually turn the axis to generate a maker pulse.
3. Click OK to close the message box and run the marker test.
4. If the test succeeds, GML Commander will display a successful test message.
5. If the test fails, check the encoder hookup and return to step 1.

Align Absolute Device

If you are using Absolute devices, you must align the absolute position of the device to correspond to the position of the axis.

Note: This selection applies only to controllers using iCODE version 3.2 and higher.

How to Align an Absolute Device

Select this to let an absolute encoder or resolver align itself to an axis' zero position, and simultaneously update the axis home position.

Be sure all external components are connected and the program, including all configured Axis and Drive data, is downloaded.

ATTENTION: Failure to download can cause uncontrolled motion when performing controller diagnostic routines.

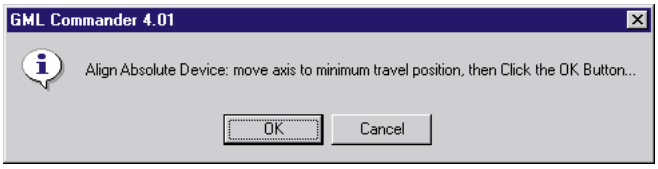
Then, do the following:

1. In the Feedback page of the Configure Axis Use dialog box, select an absolute Transducer Type.

2. In the Homing page of the Configure Axis Use dialog box, select an absolute Procedure type.

3. In the Hookups page of the Configure Axis Use dialog box, select Align Absolute Device.

4. In response to a GML Commander message box, move the axis to its minimum travel position, then select OK to close the message box.



5. The alignment procedure reads the absolute position of the transducer. If no encoder noise or loss faults have occurred during the procedure (about 2 seconds long), the routine

   - aligns the absolute device to the zero position of the axis, and
   - updates both the working and power-up home position values, and changes the Position setting in the Homing page of the Configure Axis dialog box.

Drive Offset

This tests the setup of the drive for IMC S Class Compact and Basic/ Integrated controllers that are configured for Velocity drive interface types in either the Servo or Motor/Drive page of the Configure Axis Use dialog box.

Note: For IMC S Class 1394/1394 Turbo controllers, the drive offset test is included in the motor/encoder test.

The Drive Offset Test

Be sure all external components are connected and the program, including all configured Axis and Drive data, is downloaded.

ATTENTION: Failure to download can cause uncontrolled motion when performing controller diagnostic routines.

Then, do the following:

1. In the Hookups page of the Configure Axis Use dialog box, select the Drive Offset button. The controller tests for drift, and displays a message box containing the cumulative drift.
2. Go to the drive and manually adjust the drive offset potentiometer, so that the drive is as near to perfectly still as possible.
3. In GML Commander, select OK. The message box closes, and GML Commander completes the two-step test by internally performing its own drive offset compensation.

Note: The Drive Offset test applies only to a Velocity Drive Interface (as set in the Servo page).

For IMC S Class 1394/1394 Turbo controllers, GML Commander includes this test as part of the Motor/Encoder test.

Pulse Magnitude

Type the voltage to be generated for the battery box test, from 0.0 to 9.999 Volts.

Pulse Duration

Type the duration of the voltage pulse for the battery box test, from 1 to 999.0 seconds.

Pulse Type

Select the pulse type for the battery box test, from the drop-down menu.

Battery Box Test

The Battery Box test helps adjust the response and gain potentiometers on most velocity loop servo drives (amplifiers), by sending test signals directly to the servo amplifier.

How to perform the Battery Box test

Be sure all external components are connected and the program, including all configured Axis and Drive data, is downloaded.

ATTENTION: Motion does occur during the Battery Box Test.

ATTENTION: Failure to download can cause uncontrolled motion when performing controller diagnostic routines.

Then, do the following:

1. Connect an oscilloscope to the tach (or velocity output) of the motor
2. In the Hookups page of the Configure Axis Use dialog box, type entries in the Pulse Magnitude and Pulse Duration fields, and select a Pulse Type from the drop-down list.
3. Select the Battery Test button.
4. View the battery test results in the oscilloscope. This displays the resulting velocity profiles and lets you evaluate the adjustments. (Refer to your servo drive manual for recommended drive potentiometer settings.)
5. Repeat steps 2, 3 and 4, as necessary.

Note: The Battery Box test applies only to a Velocity Drive Interface (as set in the Servo page).

Tune Servo

The Tune Servo page performs tuning routines that determine the proper settings for a servo axis' servo loop parameters. These parameters include: Proportional (P), Velocity (V), Integral (I) and Velocity Feedforward (F) gains, as well as Maximum Speed, Maximum Acceleration and Maximum Deceleration values. When tuning is complete, GML Commander automatically sets these values in the Gains and Dynamics pages of this dialog box.

Usually, these servo loop parameters need to be tuned only once, when the motion controller is first integrated onto the machine, or when the machine is being commissioned at start-up. However, if the load on any axis changes significantly, or if the motor or servo drive (amplifier) is replaced for any reason, re-tuning these servo loop parameters may be necessary.

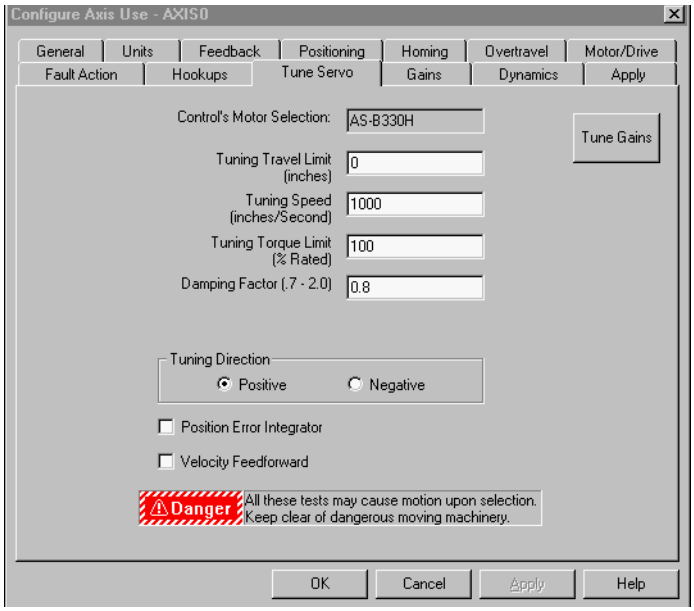
ATTENTION: All tuning tests can cause motion upon selection. Keep clear of dangerous moving machinery.

Tuning Your 1394

If you are using a 1394 controller, use this procedure to tune your drive, axes, and motors. When you enter values in the fields in this dialog box and select Tune Gains, GML Commander automatically sets values in the Gains and Dynamics dialog boxes.

Note: If necessary, you can change the default start-up values on the Gains and Dynamics dialog boxes while you are offline or fine-tune the working values for a single, online test run.

1. Select the Tune Servo tab. A page similar to the following appears:



2. Make entries in the following fields:

Field	Description
Control's Motor Selection	The value from the *Motor ID* field on the Motor/Drive dialog box appears.
Tuning Travel Limit	Type the number of units that is a safe travel distance given the practical travel limits of your application. **Note:** The greater the distance, the better for tuning.
Tuning Speed	Type the speed that the axis must reach during tuning.
Tuning Torque Limit	Type a maximum motor torque value from 0 to 300%.

Field	Description
Damping Factor	Type a value between 0.7 and 2.0 to compute the loop gains. **Note:** We recommend a default value of 0.8 to provide good gain values for most systems.
Position Bandwidth	Type value in hertz that the position loop bandwidth needs for your application. **Note:** The position loop responds more rapidly as the bandwidth is increased.

3. In the *Tuning Direction* area, make entries in the following fields:

Field	Description
Positive	Tune in a positive axial direction.
Negative	Tune in a negative axial direction

4. Make entries in the following fields:

Field	Description
Position Error Integrator	Calculates and enters a value for the *Integral Gain field* in the Gains dialog box.
Velocity Feedforward	Calculates a value for the *Feedforward gain* field in the Gains dialog box.

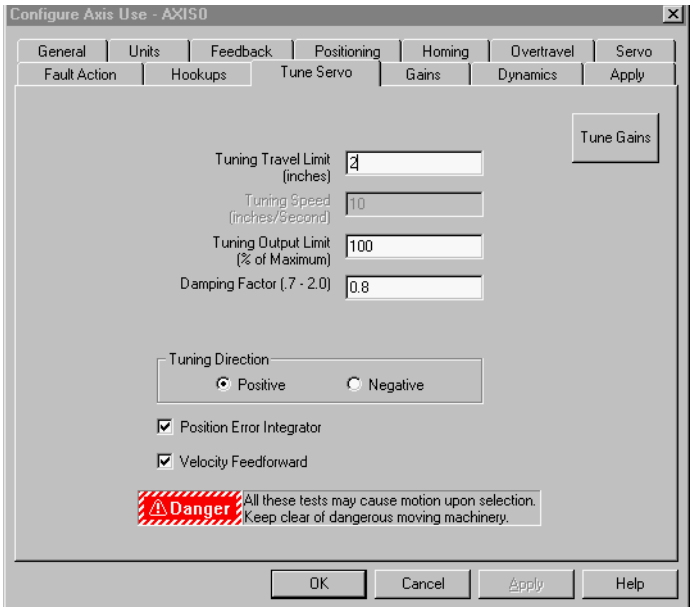
5. Select **Tune Gains**. The proportional and velocity gains are entered in the Gains dialog box.

Tuning Your Compact

If you are using a Compact, use this procedure to tune your drive, axis, and motor. When you enter values in the fields in this dialog box and select Tune Gains, GML Commander automatically sets values in the Gains and Dynamics dialog boxes.

Note: If necessary, you can change the default start-up values on the Gains and Dynamics dialog boxes while you are offline or fine-tune the working values for a single, online test run.

1. Select the Tune Servo tab. A page similar to the following appears:



2. Make entries in the following fields:

Field	Description
Tuning Travel Limit (inches)	Type the number of units that provide a safe travel distance given the practical travel limits of your application. **Note:** The greater the distance, the better for tuning.
Tuning Speed	Type the speed that the axis must reach during tuning.
Tuning Output Limit	Type a voltage output limit, as a percentage of the value entered in the *Control Output Limit Value* field in the Servo dialog box.
Damping Factor	Type a value between 0.7 and 2.0. **Note:** We recommend a default value of 0.8 to provide good gain values for most systems.

3. In the *Tuning Direction* area, make entries in the following fields:

Field	Description
Positive	Tune in a positive axial direction.
Negative	Tune in a negative axial direction

4. Make entries in the following fields:

Field	Description
Positive Error Integrator	Calculate and enter a value in the *Integral Gain* field in the Gains dialog box.
Velocity Feedforward	Calculates value for the *Feedforward gain* field in the Gains dialog box

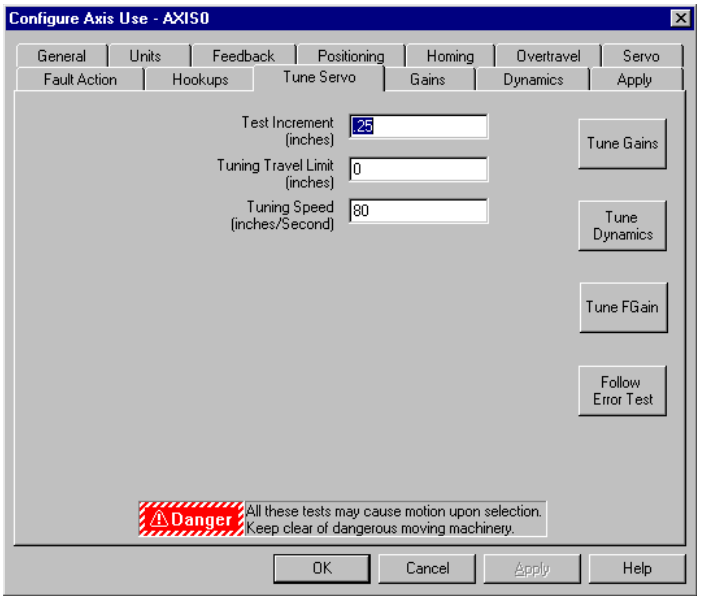
5. Select **Tune Gains**. The proportional and velocity gains are entered in the Gains dialog box.

Tuning Your Integrated/Basic

If you are using an Integrated or Basic controller, use this procedure to tune the axis motor and drive. When you enter values in the fields in this dialog box and select Tune Gains, GML Commander automatically sets values in the Gains and Dynamics dialog boxes.

Note: If necessary, you can change the default start-up values on the Gains and Dynamics dialog boxes while you are offline or fine-tune the working values for a single, online test run.

1. Select the Tune Servo tab. A page similar to the following appears:



2. Make entries in the following fields:

Field	Description
Test Increment	Type the distance for the test. **Note:** The recommended distance is ¼ of an encoder revolution.
Tuning Travel Limit	Type the number of units that provides a safe travel distance given the practical travel limits of your application. **Note:** The greater the distance, the better for tuning.
Tuning Speed	Type the speed that the axis reach during tuning.

3. Select **Tune Gains**. The proportional, velocity, and integral gains are entered in the Gains dialog box.

4. Select **Tune Dynamics**. The maximum speed, acceleration, and deceleration and the error tolerance are entered in the Dynamics dialog box.

5. Select **Tune Fgain**. The feedforward gain is calculated and entered in the Gains dialog box.

6. Select **Follow Error Test**. The following error test measures the position error at three critical points during a trapezoidal profile move at maximum velocity and maximum acceleration and deceleration. Position error while the axis is moving is called following error. At the end of the move the axis will move back to the starting point and the three error values will be displayed. Ideally, all three of the error values should be close to zero. Unfortunately, there are trade-offs that must be carefully weighed to achieve optimal system performance.

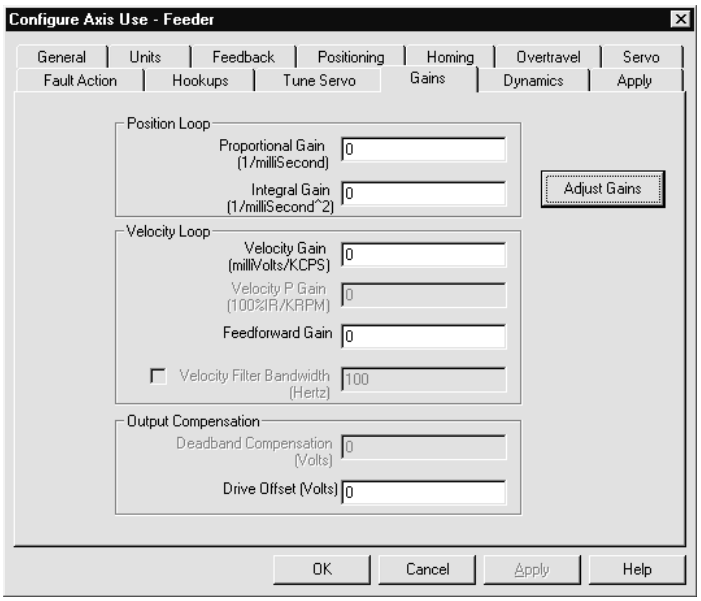
Defining Gains

The Defining Gains page is for:

- Manually tuning your application programs' power-up Gains values. You can do this instead of accepting the Gains automatically generated by GML Commander in the Tune Servo page. (These settings changes are made off-line. They take effect, and become working values, only after you download them to the controller.)

Important: It is strongly advised that you accept the gain values automatically generated by GML Commander.

- Opening the Adjust Gains page, where you can make real-time, on-line adjustments to the controller's working Gains values.



Manually configuring the Gains page requires you to make entries in the following fields:

Proportional Gain

A value, in position units per millisecond.

Note: Increasing this number decreases response time and increases the stiffness of the servo system. If the value is too high, the system becomes unstable.

Integral Gain

A value, in position units per millisecond squared.

Note: Increasing this number improves positioning accuracy of the servo system. If the value is too high, the system becomes unstable.

Velocity Feedforward Gain

The number of position units that allows the following error (position error while the axis is moving) of the system due to velocity, to be reduced to near zero.

Velocity Gain

A value, in millivolts/counts per millisecond.

Increasing this value results in smoother motion, enhanced acceleration, reduced overshoot, and greater system stability. Too high a value leads to high frequency instability.

Important: For IMC S Class 1394/1394 Turbo and Compact controllers only.

Velocity P Gain

A value, in % of I Rated per thousand RPM, to change the digital drive velocity loop Proportional gain.

Important: For IMC S Class 1394/1394 Turbo controllers only.

Velocity Filter Bandwidth

A value, in Hertz, limiting the DSP sub-system's response to changes in commanded velocity for the axis.

Lower values provide for smoother system operations. However, if a value is too low, a phase error can result.

Important: For IMC S Class 1394/1394 Turbo controllers only.

Deadband Compensation

For a torque (current) loop servo drive, the voltage value equal to the amount of controller voltage command that is too small for the drive to respond to.

Important: For IMC S Class Compact and Basic/Integrated controllers only.

Drive Offset

A value, in volts, sufficient to offset any drive voltage output error.

Important: For IMC S Class Compact and Basic/Integrated controllers only.

Adjusting Gains Values for the Servo Axis

The Adjust Gains dialog box is accessible only if the start-up (offline) gain values and the working (online) gain values are identical.

ATTENTION: The Up and Down buttons immediately affect the control's gains. Some gains values can cause unpredictable motion. Keep clear of dangerous moving machinery.

Equalizing the Working and Set-up Gains Values

You must be online to set the working and the setup gain values equal to each other. To set the values, do one of the following:

- On the Apply page of the Configure Axis Use dialog box, select **Upload** or **Download** (depending upon the version of the axis use configuration you want to use).
- On the General page of the Control Options dialog box, select **Download Axis/Drive**. The current application program is downloaded.

Adjusting Gains

You can use the Adjust Gains dialog box to:

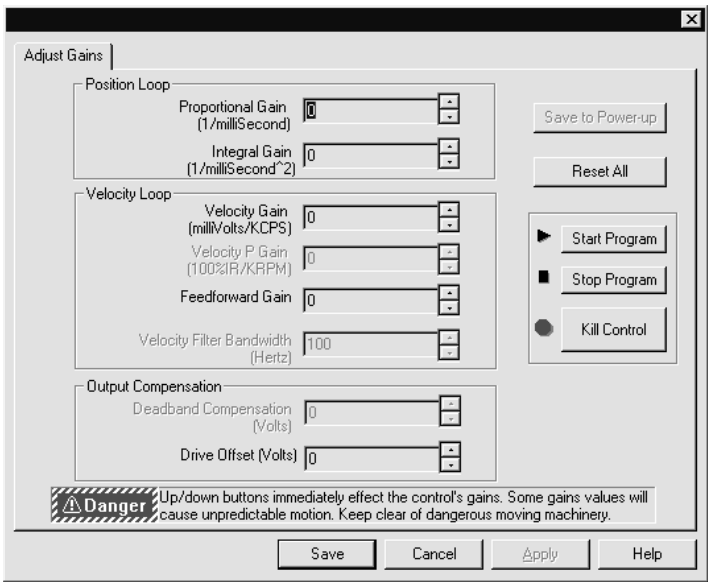
- Make incremental changes to working gain values, start-up gain values, or both.
- Start, stop, or kill the program, as needed, as you fine-tune the gain values.

These digitally-set gain values provide software control over the servo dynamics and allow the servo system to stabilize.

You make all changes to gain with the up and down arrows on each field's value list. A change to a gain value is made in small increments (thousandths or ten-thousandths of a unit) to protect material and machinery from sudden and potentially harmful jolts. When you make a change to any gain value, the new value immediately becomes the new working value for that gain.

To adjust gain values:

1. Select **Adjust Gains** from the Gains page of the Configure Axis Use dialog box. A dialog box similar to the following appears:



2. In the *Position Loop* area, make entries in the following fields:

- Proportional Gain – select the up or down arrows to change the response times to increase the stiffness of the servo system.

Note: If the proportional gain is too high, the system is unstable. If the proportional gain is too low, the system is loose or sloppy.

- Integral Gain – Select the up or down arrows to improve the steady-state positioning performance of the system.

Note: Increase the integral gain to increase the ultimate positioning accuracy of the system. If the integral gain is too high, the system becomes unstable.

3. In the *Velocity Loop* area, make entries in the following fields:

- Velocity Gain – Select the up or down arrows to change the speed of velocity changes.

Note: When using a torque (current) loop servo amplifier, the synthesized velocity loop provides damping without requiring an analog tachometer. Increased V gain results in smoother motion, enhanced acceleration, reduced overshoot, and greater system stability.

- Velocity P Gain (100%IR/KRPM) – Select the up or down arrows to change the digital drive velocity loop P gain.
- Feedforward Gain – Select the up or down arrows to reduce the error of the system, due to velocity (position error while the axis is moving), to nearly to zero.
- Velocity Filter Bandwidth – Select the up or down arrows to change the value of the digital drive's velocity loop bandwidth.

4. In the *Output Compensation* area, make entries in the following fields:

- Deadband Compensation – Select the up or down arrows to change the voltage of the deadband compensation.

Note: It is generally unnecessary to alter the value calculated from an auto-tuning procedure.

Note: For a velocity loop servo amplifier, adjust the drive-offset compensation.

Note: For torque or dual servo drive interface type only.

- Drive Offset – Select the up or down arrows to change the drive offset for this axis.

Note: Note: For Compact, Integrated, and Basic controllers only.

5. Saving Changes

- Save to Power-up – Saves changed working values as the power-up (or start-up) values in the controller
- Save – Saves changed working values as the power-up values in the Gains dialog box

Note: This does not change the power-up values in the controller until you download with axis/drive data selected, download via the Apply page, or press the Save to Power up in the Adjust Gains page.

- **Save then** select **OK** in the Configure Axis Use dialog box**–** Saves changed working values as offline power-up values
- **Cancel** – Not save the changes

Note: Generally, you will not need to modify the self-tuned parameters for axis gains.

Running your Program

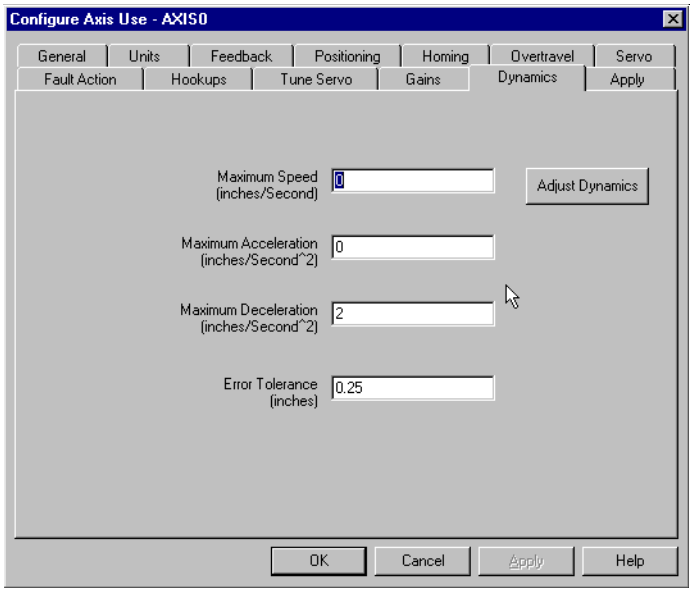
To control your program from the Adjust Gains page, use the following:

- **Start Program –** Starts the application program in the controller.
- **Stop Program –** Stops the application program in the controller with the feedback remaining enabled and motion stopping at 100% of the deceleration rate.
- **Kill Control –** Kill the control with the feedback disabled and motion stopping immediately on all axes.
- **Reset All –** Resets the values on this page to the controller's power-up (start-up) values.

Defining Dynamics

Use the Defining Dynamics page to:

- Manually tune your application program's power-up Maximum Speed, Acceleration and Deceleration values. This is done instead of accepting the values automatically generated by GML Commander in the Tune Servo page. (These settings changes are made off-line. They take effect, and become working values, only after you download them to the controller.)
- Set Error Tolerance.
- Open the Adjust Dynamics page, where you can make real-time, on-line adjustments to the controller's working values for Maximum Speed, Acceleration and Deceleration, as well as Error Tolerance.



The Dynamics page can be manually configured by making entries in the following fields:

Maximum Speed

A value, in position units per second, to set the maximum speed for this axis.

Maximum Acceleration

The value, in position units per second squared, that sets the maximum acceleration rate for this axis.

Maximum Deceleration

The value, in position units per second squared, for setting the maximum deceleration rate for this axis. When a motion block asks you to enter a value as a % of Max, it is referring to the three Maximum value fields set in this page.

Error Tolerance

A value, in position units, specifying the greatest difference, between actual position and command position, the controller will tolerate before indicating a Position_error_fault . In the event of a position error fault, GML Commander will take the action you selected in the Fault Action page.

Adjusting Dynamics Values for the Servo Axis

The Adjust Dynamics dialog box is accessible only if the start-up (offline) dynamics values and the working (online) dynamics values are identical.

ATTENTION: The Up and Down buttons immediately affect the control's dynamics. Some dynamics values can cause unpredictable motion. Keep clear of dangerous moving machinery.

Equalizing the Working and Set-up Dynamics Values

You must be online to set the working and the setup Dynamics values equal to each other. To set the values, do one of the following the following:

- On the Apply page of the Configure Axis Use dialog box, select **Upload** or **Download** (depending upon the version of the axis use configuration you want to use).
- On the General page of the Control Options dialog box, select **Download Axis/Drive**. The current application program is downloaded.

Adjusting Dynamics

You can use the Adjust Dynamics dialog box to:

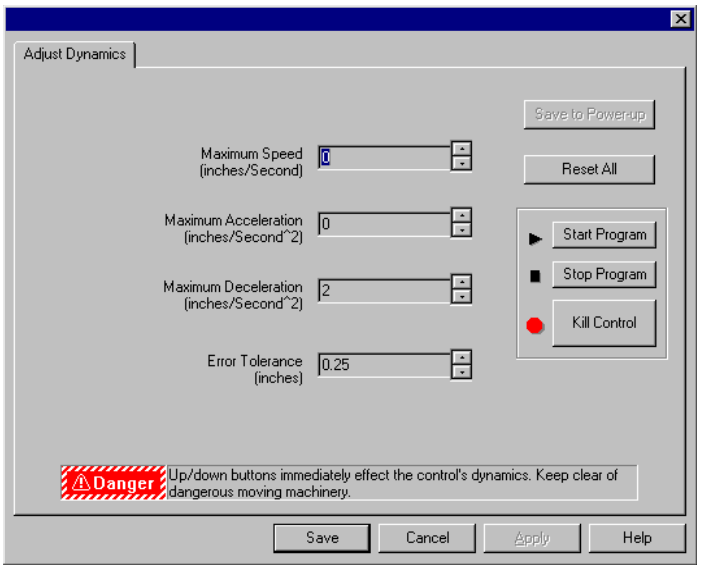
- Make incremental changes to working dynamics values, start-up dynamics values, or both.
- Start, stop, or kill the program, as needed, as you fine-tune the dynamics values.

These digitally set dynamics values provide software control over the servo dynamics and allow the servo system to stabilize.

You make all changes to dynamics with the up and down arrows on each field's value list. A change to a dynamics value is made in small increments (thousandths or ten-thousandths of a unit) to protect material and machinery from sudden and potentially harmful jolts. When you make a change to any dynamics value, the new value immediately becomes the new working value for that dynamics.

To adjust dynamics values:

1. Select **Adjust Dynamics** from the Dynamics page of the Configure Axis Use dialog box. A dialog box similar to the following appears:



2. Make entries in the following fields:

   - **Maximum Speed** – Select the up or down arrows to change the maximum speed for this axis.
   - **Maximum Acceleration** – Select the up or down arrows to change the maximum positive acceleration rate for this axis.
   - **Maximum Deceleration** – Select the up or down arrows to change the maximum deceleration rate for this axis.

- **Error Tolerance** – Select the up or down arrows to change how much error (relative to the position of the axis) the controller tolerates before indicating a position error fault.

3. Saving Settings

- **Save to Power-up** – Saves changed working values as the power-up (or start-up) values in the controller
- **Save –** Saves changed working values as the power-up values in the Dynamics dialog box

Note: This does not change the power-up values in the controller until you download with axis/drive data selected, download via the Apply page, or press the Save to Power up in the Adjust Dynamics page.

- **Save** then select OK in the Configure Axis Use dialog box – Save changed working values as offline power-up values.
- **Cancel –** Not save the changes

Note: Generally, you will not need to modify the self-tuned parameters for axis dynamics.

Running your Program

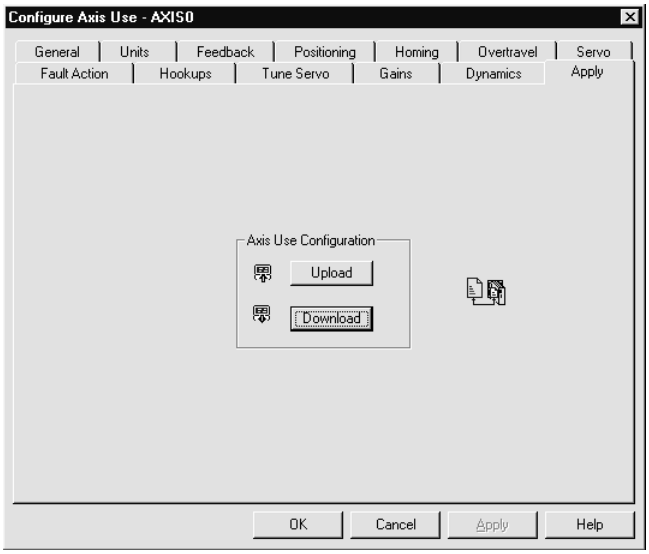
To control your program from the Adjust Dynamics page, do the following:

- **Start Program –** Starts the application program in the controller.
- **Stop Program –** Stops the application program in the controller with the feedback remaining enabled and motion stopping at 100% of the deceleration rate.
- **Kill Control –** Kills the control with the feedback disabled and motion stopping immediately on all axes.
- **Reset All –** Resets the values on this page to the controller's start-up values.

Applying Axis Configuration Changes

The Applying Axis Configuration Changes is used to:

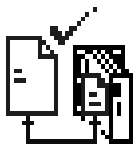
- Determine if the axis configuration in the controller is the same as the axis configuration in GML Commander.
- Upload axis configuration from the controller to GML Commander, or download axis configuration from GML Commander to the controller, without having to close the Configure Axis Use dialog box.



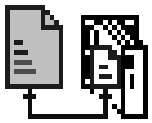
Use the features of the Apply page as follows:

Controller Icon

The icon in the right-center of the dialog box indicates whether axis configuration in the controller is the same as in GML Commander. If the page is white with a blue checkmark above it, as shown below, the axis configuration in the controller and in GML Commander are the same.



If the page is grayed-out and there is no checkmark, as shown below, the axis configuration in the controller and in GML Commander are different.



Upload button

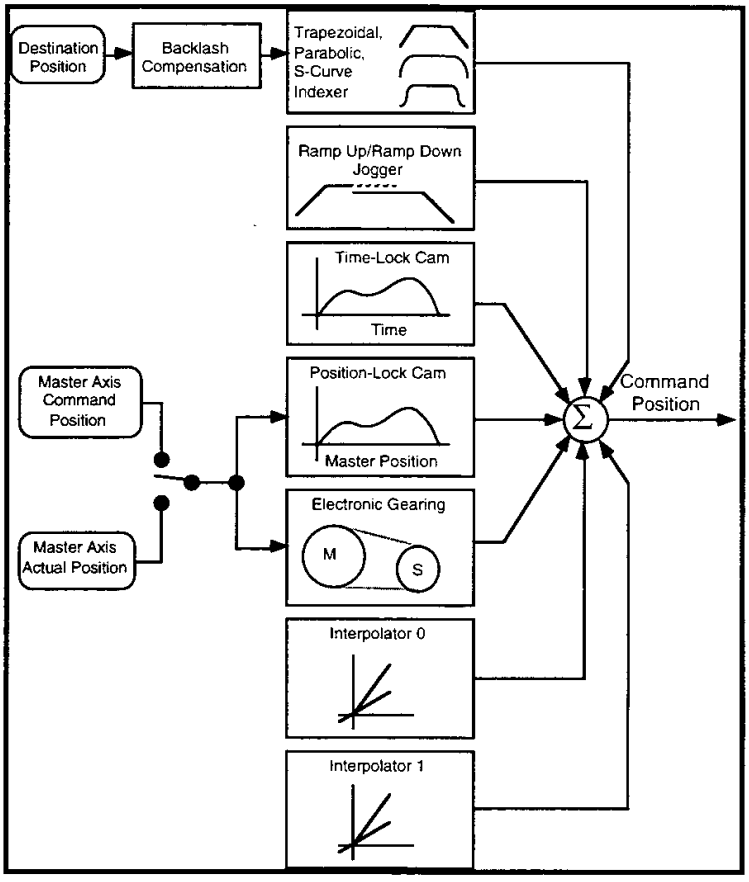
If the axis configurations, in the controller and in GML Commander, are different, select Upload to load the axis configuration from the controller to GML Commander.

Download button

If the axis configurations in the controller and in GML Commander are different, select Download to load the axis configuration from GML Commander to the controller.

Using the Imaginary Axis

The imaginary axis (available only in firmware 2.2 and later) is an internal axis with no connection to a drive or encoder. It is used as a master axis that other real axes are synchronized to when no physical or virtual axis is available. As such, it can be thought of as a built-in virtual axis not requiring AxisLink. As shown below, the output of the imaginary axis is its command position—it has no actual position.



As shown previously, all the motion generating functions of physical servo axes are available for the imaginary axis; it just cannot actually move a motor. Each motion controller can have one imaginary axis defined.

Configuring the Imaginary Axis

Like all other axes, the imaginary axis must be configured before it can be used. As with any axis, configuring an imaginary axis is a two-step process.

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select the **Axes** tab. (In Novice Mode, select **Next** to open the Axes page.)
4. Select **Imaginary**. (Optionally, you can name the imaginary axis as appropriate.) The default name, IMAGINARY, appears next to the check box.
5. Select **OK**.
6. From the menu bar, select **Configure**. The Configure menu appears.
7. Select **Axis Use**. The Axis Use axis list appears.
8. Select **IMAGINARY** (or the name that you assigned to the imaginary axis). The Configure Axis Use-IMAGINARY dialog box appears.
9. Make the appropriate settings. (See the *Configuring Axis Use* chapter of this manual for field definition information on the setup menus used to configure an axis.)
10. Select **OK**.

Like other axes, you can configure the imaginary axis as either a rotary or linear axis with its own position units, conversion constant and (if the axis is rotary) unwind value. Select both the move and jog profiles, and set the maximum speed, maximum acceleration, and maximum deceleration values for the imaginary axis. Needless to say, hookup diagnostics and self-tuning are not available for an imaginary axis. See the *Setup* section of the *Installation and Setup* manual for your motion controller for information on using the setup menus to configure the imaginary axis.

The conversion constant for the imaginary axis is essentially arbitrary, but does affect the smoothness of gearing and position-lock cams that use the imaginary axis as their master. For best results, set the position units and conversion constant for the imaginary axis to the same values used for the physical axis that is to be slaved to the imaginary axis.

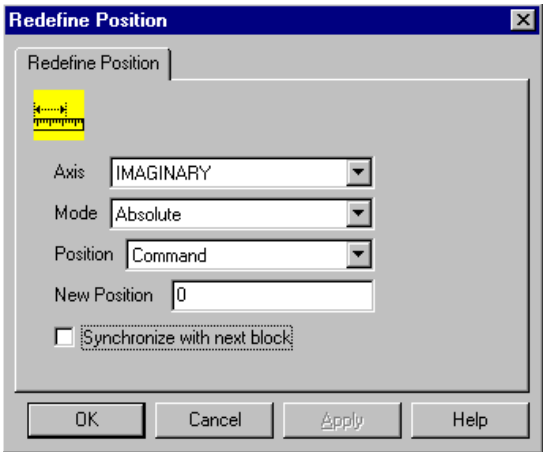
Using the Imaginary Axis in GML

All motion blocks normally used for physical servo axes (Move Axis, Jog Axis, Position-Lock Cam, etc.) can also be used with imaginary axes, with the following exceptions:

- Home Axis
- Direct Drive Control
- Watch Control - Arm and Disarm Registration
- On Watch - Wait for Registration

Because feedback, drive, or registration inputs are not associated with an imaginary axis, these blocks do not apply.

To home the imaginary axis, use the Redefine Position block to set the command position of the imaginary axis to the desired value. For example, the following selections home the imaginary axis to zero.



When used with an imaginary axis, both of the following blocks, use the command position (versus the actual position) of the imaginary axis':

- Watch Control - Arm Watch Position block
- On Watch - Wait for Position Event block

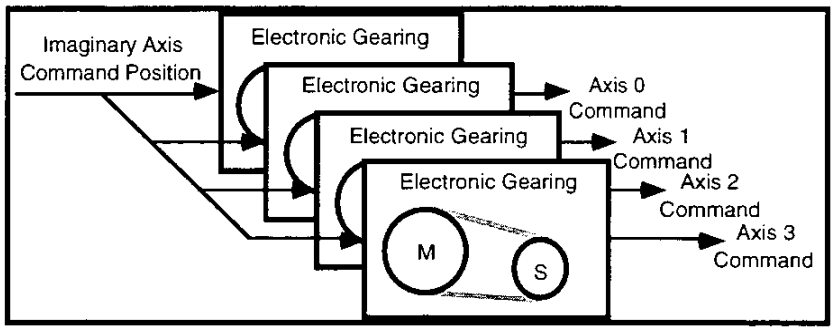
This is because the imaginary axis has no actual position. By contrast, when used on servo and master-only axes, these blocks use the axis' actual position.

While all of the appropriate motion status variables (Jog_status, Accel_status, etc.) apply also to the imaginary axis, the imaginary axis never indicates an axis locked status. Thus Lock_status for the imaginary axis is always 0 (Unlocked) and Axis_status is always greater than 0.

The normal axis status for an enabled imaginary axis with no commanded motion is unlocked (Axis_status = 1). In addition, no axis faults are ever active for the imaginary axis. Thus Axis_fault and the other axis-specific fault variables are always 0 for this axis. Average_Velocity for an imaginary axis is always zero. Use Command_Velocity for the velocity of an imaginary axis. See the *System Variables* chapter in this manual for more information on these variables.

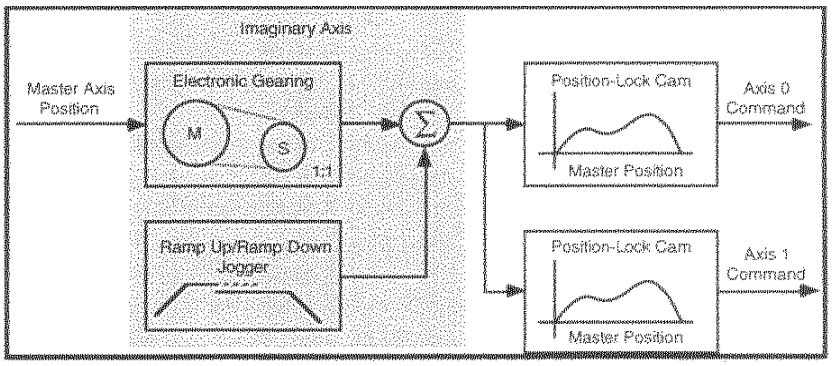
Imaginary Axis Applications

You use the imaginary axis to synchronize multiple physical axes together using an independent master. Thus, all physical axes can be used as slaves, instead of using just one as the master for the others. This results in closer synchronization between the physical axes since they are all following the same command, as shown below.



To move all four physical axes together, execute a move, jog, or time-lock cam on the imaginary axis.

The imaginary axis also allows manual jogging of multiple synchronized axes without losing synchronization. In this type of application, the imaginary is both a slave and a master axis. The imaginary axis is geared at a 1:1 ratio to the real (physical or virtual) master and the imaginary axis' command position is then used as the master for the physical slave axes, perhaps executing position-lock cams, as shown in the following diagram.



By inserting the imaginary axis between the true master and slave axes, gearing on the imaginary axis can be turned off to uncouple all the slaves from the master simultaneously without stopping each position-clock cam. In addition, by jogging or moving the imaginary axis, you can move the slaves together without loss of synchronization between them. This allows manual tweaking or offsetting of a synchronized process involving many axes.

The imaginary axis also makes it possible for two moves to occur simultaneously on the same axis (one move can be superimposed on top of another move). This technique is excellent for implementing continuous slip compensation on a material feeding axis that uses an absolute move to feed the material and a second encoder riding on the material to measure the slip.

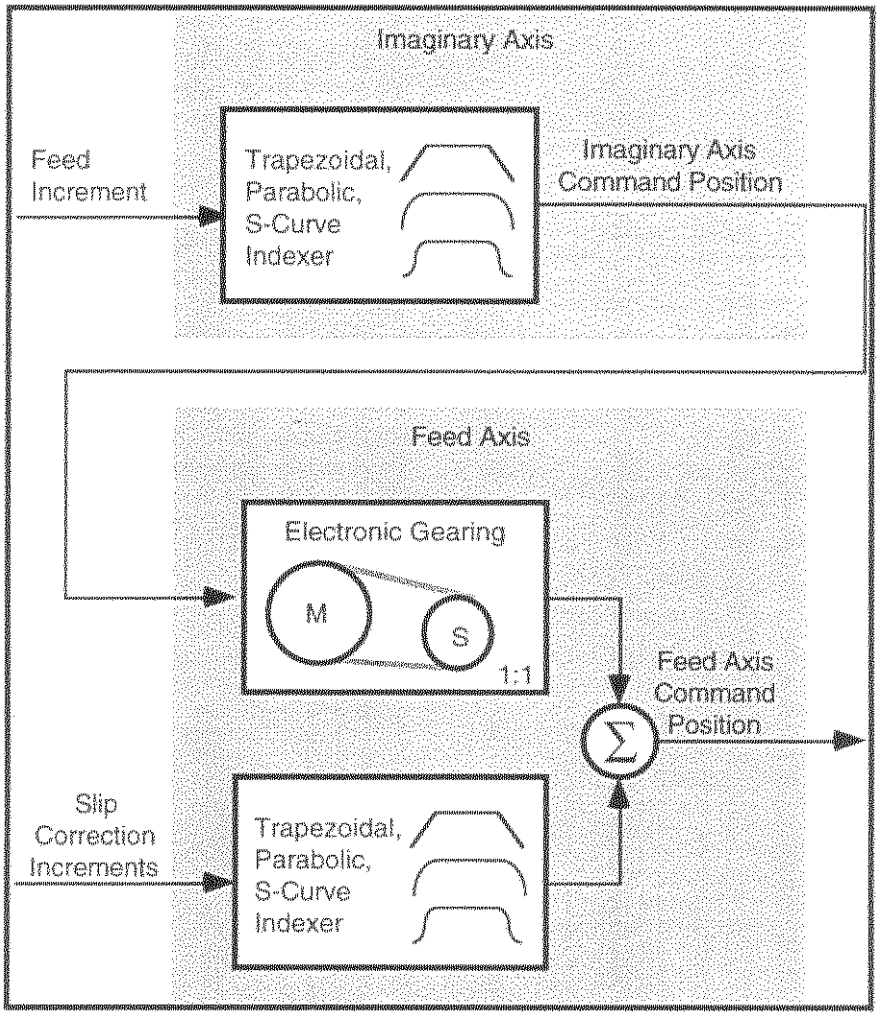
Because a move is used to implement the feed, another move cannot ordinarily be superimposed on the feeding move to do continuous slip compensation. While the endpoint of the feed move could be changed while the move is in progress using another move command, this approach has a number of disadvantages:

- To avoid the axis hopping into final position, the corrections can only be made while the axis is moving. Thus, any slip that occurs after the last correction but before the axis stops is not compensated.

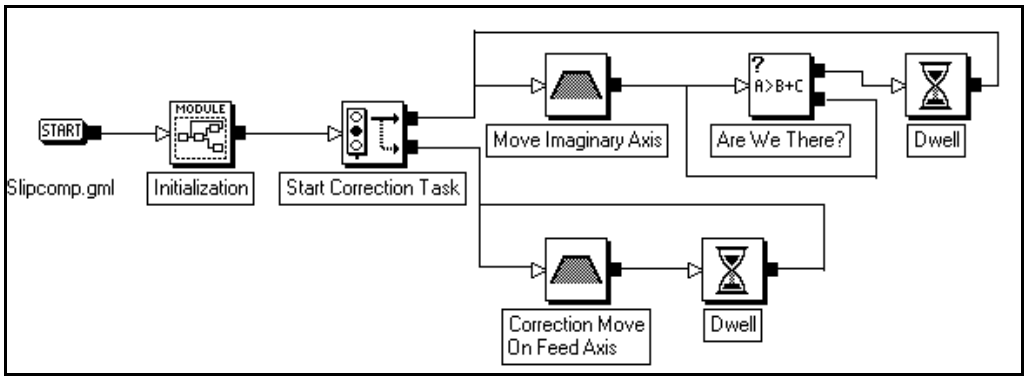
- This technique works with trapezoidal profile moves, and with S-curve profile moves using version 3.6 (or higher) firmware. You cannot change the endpoint of a parabolic move, or an S-Curve move using version 3.5 (or lower) firmware while the axis is decelerating.
- The position of the feed axis is polluted by the corrections, and the actual (and command) position of the axis at the end of the feed is different from its true physical position by an amount equal to the compensated slip. This makes back-to-back absolute feeds virtually impossible.

Using the imaginary axis to superimpose two moves on the feed axis overcomes these disadvantages.

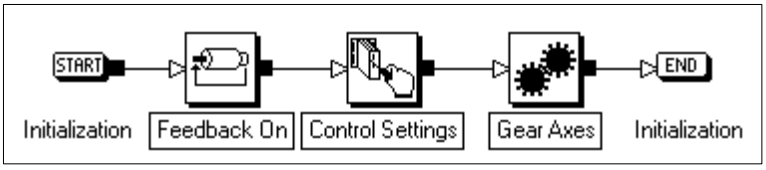
To implement continuous slip compensation using the imaginary axis, gear the feed axis to the imaginary axis at a 1:1 ratio and move the feed axis indirectly by moving the imaginary axis. You can now make corrections as incremental moves to the real (physical) feed axis based on the position of the material encoder and they are superimposed on the gearing (really another move) motion from the imaginary axis, as shown in the following diagram.



Using the imaginary axis allows two moves to be superimposed on one another in the same way that a move is superimposed on gearing. A GML diagram that implements this technique is shown in the following diagram and included with the GML distribution software in the Samples directory, as SLIPCOMP.GML.



The Initialization module (below) enables feedback, reads the position lock tolerance of the feed axis into the Lock_Tol_0 user variable (more on this later) and gears the feed axis to the imaginary axis at a 1:1 ratio. Once this gearing is established, you can move the feed axis either directly or move the imaginary axis.



The main task feeds a given amount of material using an incremental move of the imaginary axis and waits for completion. However, since corrections are continuously being applied directly to the feed axis by the correction task, the feed axis can still be moving (correcting) when the imaginary axis is done. Therefore, use an On Expression block to implement a custom “wait for axis lock” function in GML. This block evaluates the following expression:

Absolute_value(Command_Position_Imaginary -
Actual_Position_Material) < Lock_Tol_0

This expression compares the absolute value of the difference between the actual position of the material encoder and the command position of the imaginary axis (the true position error in this system) to the Lock_Tol_0 user variable. Lock_Tol_0 is the position lock tolerance parameter of the feed axis, set in the Initialization module. By using the position lock tolerance parameter of the physical feed axis for this comparison, the feed axis responds to changes in its position lock tolerance just as it would if it were being moved directly. After a dwell, the feed is repeated indefinitely.

The correction routine is run as its own task (task 1), independent of the actual feeding motion. A direct incremental move of the feed axis equal to the following:

Command_Position_Imaginary - Actual_Position_Material

This expression corrects for any slip of the feed axis. This move is executed continuously based on a timer. Select Wait for Completion in this move block to guarantee that one correction move finishes before the next move starts. The value of the timer determines the frequency of the correction moves and must be set empirically based on the dynamics of the application. In some applications, no dwell may be necessary—the correction moves can be executed back-to-back using only Wait for Completion. By specifying the velocity of the correction moves as a percentage of the speed of the imaginary axis, you can make the corrections to occur faster as the line speeds up.

The correction routine runs all the time—even when the axis is stopped—and thus the corrections are applied continuously as the axis slips rather than all at the end of the feed. Be aware, however, that using the imaginary axis increases the CPU Utilization and running the correction routine as a separate task reduces the throughput of the main program. See the *CPU Utilization* chapter in this manual for more information.

Note that the imaginary axis must be set up identically to the feed axis for this technique to work properly. In addition, all axes must use the same user units to allow cross-axis position computations. In SLIPCOMP.GML, Axis 0 is the feed axis and Axis 1 is the material encoder. The imaginary axis is the imaginary axis.

While the command position of the feed axis is polluted by the corrections with this technique, the command position of imaginary axis is not. Thus, you can use the command position of imaginary axis as the true command position of the feed axis for any other calculations that may be necessary. In addition, there are no restrictions on the motion profile used for either the feed or the correction move—any of the available profiles can be used depending on the requirements of the application.

Chapter 5

Working with Blocks

To create a program or diagram in GML Commander, you place blocks representing program functions between the following two blocks:

These two blocks are always present in the diagram and cannot be deleted. You add blocks to the Diagram to Window to represent the actions that are to be performed. You then connect the blocks to show the flow of program functions. You can move the blocks around and change their connections, which changes program flow. You can simplify complex diagrams by grouping (encapsulating) several blocks together, then giving the module a name that describes the group's function.

To make working with blocks easier, we recommend that you:

- Make your block connections flow from left to right and from top to bottom.
- Limit the number of blocks in the workspace to ten or fewer.
- Try to keep your diagram on one screen so that you don't need to scroll to see parts of it.

This chapter deals with the general information about blocks and the mechanics of placing blocks within the Diagram Window. For further information on specific blocks see one of the appropriate block chapters.

Selecting and Positioning Blocks

You can open and close palettes containing blocks that are dedicated to perform specific GML Commander functions exactly as you would open and close Window toolbars.

To select a block from an open palette and position it in a diagram:

1. Move the cursor over the desired function block in a palette.
2. Select the block by clicking the left mouse button once. The cursor changes to a pointing finger.
3. Place the cursor into position in the Diagram Window between the Start and End blocks.

4. Click the left mouse button once. The block appears in the diagram.

Note: You can also click-and-drag by clicking once on a block with the left mouse button to select it and, while holding down the left mouse button, drag it to a location in the diagram.

The Function Block Palettes

The color-coded building blocks that contain GML Commander functions are the graphical elements you use to create a diagram. These blocks appear beneath the menu bar in logical groups called palettes. You can position palettes anywhere on the screen in the same way as Windows toolbars.

Manipulating Blocks

Most blocks and modules, including the Start and End blocks, have input and output nodes. The diagram must start at the Start block. It also must have connections drawn from an output node of one block to the input node of another. The diagram usually ends at the END block, but does not need to.

Using the Cursor

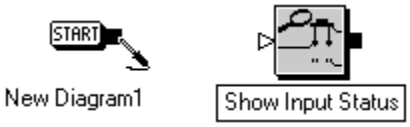
When you build a diagram, the primary cursor takes on the appearance of different tools to indicate that you can perform other diagram-editing tasks:

When the cursor appears as:	Then you:
 an arrow	Can select blocks within the diagram.
 a pointing finger	Have selected a block from the palette.
 a hand	Can move the selected block and its connections to another location within the Diagram Window.
 a soldering iron	Can connect blocks.
 a wire cutter	Can remove connections.
↔ a double arrow	Can move the green lines that connect blocks

Connecting Blocks

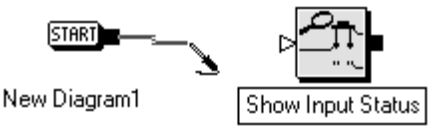
To connect blocks and modules:

1. Move the cursor to the desired output node. The cursor changes to a soldering iron.



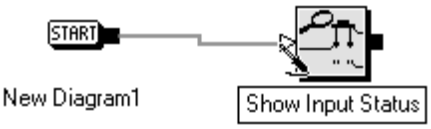
2. Press and hold the mouse button.

3. Drag the soldering to the desired input node or anywhere inside the block. As you drag, a red line representing the connection appears.



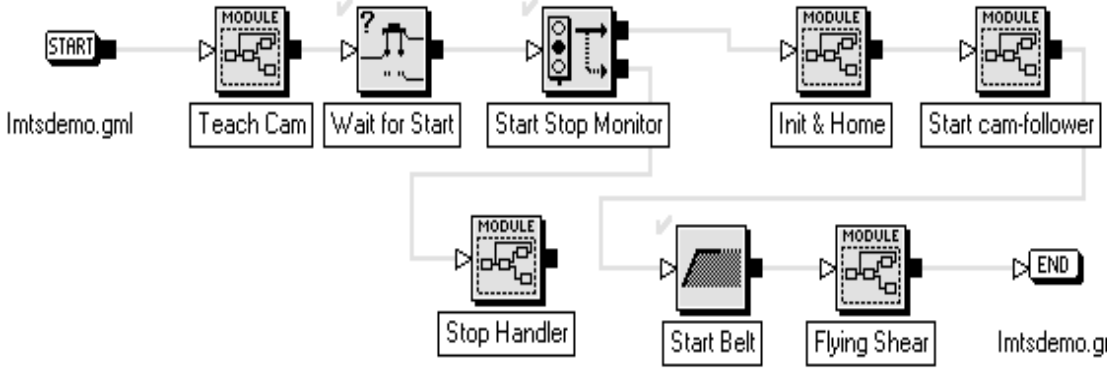
4. When the soldering iron is in the appropriate location, release the mouse button. The red line turns green.

   Important: Green lines indicate a valid connection. Therefore, if while attempting to draw a connection, the red line does not turn green, you are attempting an invalid operation.



Moving Blocks

You can drag blocks, including the Start and End blocks, anywhere in the Diagram Window. To drag a block or selection of blocks press and hold down the left mouse button on the selected blocks and drag the cursor (now in the shape of a hand) to the desired location. Then release the left mouse button. The original connection lines remain intact no matter where you move them.



Showing the Connection Lines

Use the Diagram Drawing option (select Properties from the File menu) to show connection lines:

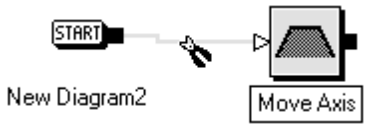
- On top of blocks to help keep track of connections in the foreground.
- Underneath blocks to show blocks in the foreground.

Disconnecting Blocks

To cut a connection:

1. Move the cursor to a connecting line. The cursor changes to a wire cutter.

   Note: You can only cut horizontal connecting lines.



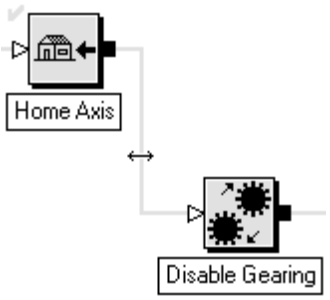
2. Click your left mouse button. The connection is cut.

Changing the Position of the Connection Lines Between Blocks

In some cases, when you have a connection that loops back to an earlier block in the diagram, you can move a vertical connection to avoid other blocks.

To move lines:

1. Place the cursor on the vertical connecting line. The cursor changes to a double arrow.



2. Hold down the left mouse button.
3. Drag the connection line to the desired position and release.

Making Changes to Your Diagram

Edit menu options generally pertain to the Diagram Window. You use this menu to make standard editing changes to your diagram, such as cut, copy, and paste.

You also use this menu to select editing features that are specific to GML Commander, such as moving and searching. These options are described in this section.

Selecting All

Use the Select All option to select all the elements in the Diagram Window. This option is often used with other editing changes to your diagram, such as aligning and spacing.

To select the entire diagram that appears in the Diagram Window:

1. From the menu bar, select **Edit**. The Edit menu appears.
2. Choose **Select All**. Every block and connection is highlighted.

Copying a Block

Use the Copy and Paste options to copy an existing block, and all of its current field and parameter settings, in a diagram one or more times.

To copy a block:

1. Select the block you want to copy.
2. From the menu bar, select **Edit**. The Edit menu appears.
3. Select **Copy**.
4. Select **Paste**. A duplicate of the block, without connection lines, appears next to the block that you originally selected.

You can paste the block as many times and to as many locations within your diagram as you need.

Swapping Blocks

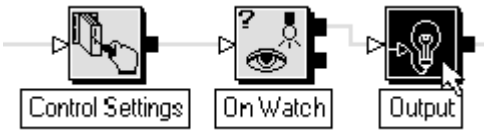
Use the Swap Blocks option to swap the location and connections of two selected blocks, even if the blocks are not connected or positioned next to each other.

In the example that follows, the Control Settings block is swapped with the Output block.

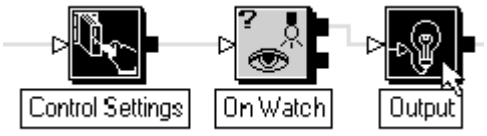
This is the current position of the blocks:

To swap two selected blocks in a diagram:

1. Select one of the blocks to be swapped by with your left mouse button. It is highlighted:

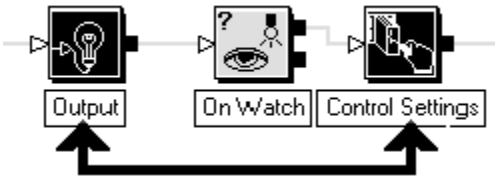


2. Select the second block to be swapped by holding down the Shift key and clicking on the second block with your left mouse button. You should now have two selected blocks.



3. From the menu bar select Edit. The Edit menu appears.

4. Select **Swap Blocks**. The blocks switch positions.



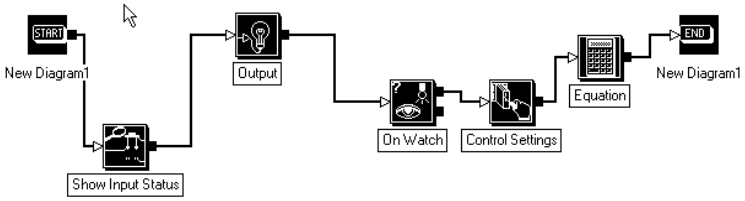
Click your left mouse button in another area of the Diagram Window to clear the selection.

Aligning Blocks

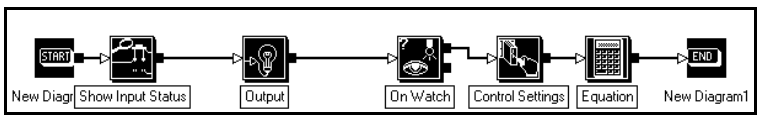
Use the Align Blocks option to align selected blocks on a vertical or a horizontal grid. The Align Blocks option affects only the blocks and inclusive connections (including the Start and End blocks) that you select.

To align an entire diagram appearing in the Diagram Window:

1. From the menu bar, select **Edit**. The Edit menu appears.
2. Choose **Select All**. All the blocks and connections in the diagram are highlighted.



3. From the menu bar, select **Edit**. The Edit menu appears.
4. Select **Align Blocks**. The selected blocks align.



5. Click your left mouse button in another area of the Diagram Window to clear the selection.

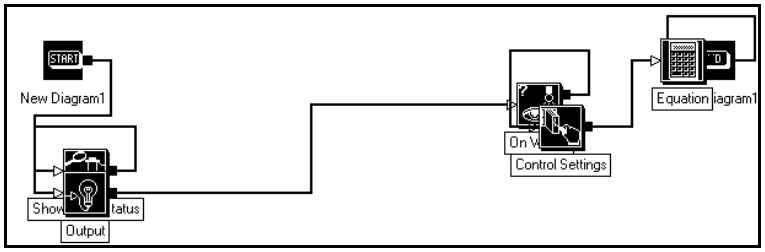
Note: You can visually align blocks and connections anywhere in the Diagram Window. See *Moving Blocks and Changing the Position of the Connection Lines Between Blocks*.

Spacing Blocks

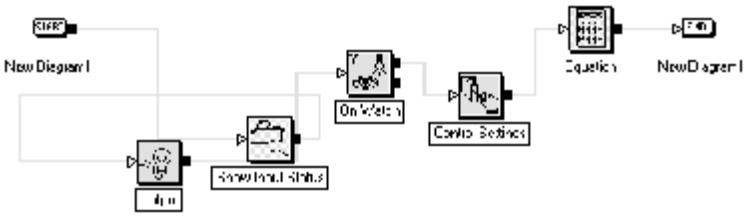
Blocks that are evenly spaced are easier to read. When a diagram becomes cluttered and blocks overlap, it is time to space the blocks so you can keep track of what you are doing.

To space blocks of an entire diagram appearing in the Diagram Window:

1. From the menu bar, select **Edit**. The Edit menu appears.
2. Choose **Select All**. The entire diagram is selected.



3. From the menu bar, select **Edit**. The Edit menu appears.
4. Select **Space Blocks**.
5. Click your left mouse button in another area. The diagram selection is cleared and the blocks are uniformly spaced.



Note: You can Align Blocks at this time for optimum readability.

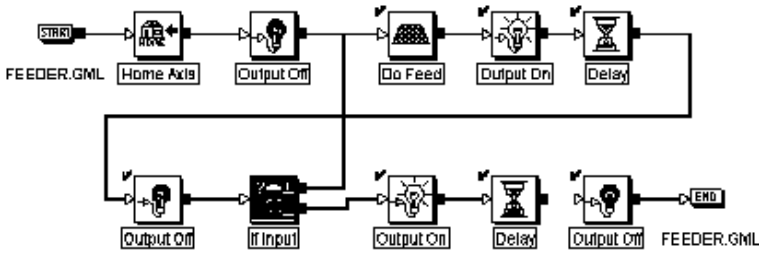
6. Click your left mouse button in another area of the Diagram Window to clear the selection.

Snapping to the Grid

You define how you place blocks in the diagram by using the Snap To Grid option in the Properties dialog box. GML Commander divides the diagram into invisible horizontal and vertical lines.

You can specify that blocks be aligned to the nearest horizontal and/or vertical grid line when placed in the diagram.

Below is an example of a diagram aligned with Snap To Horizontal. Note that blocks can flow to a additional rows if required.



Accessing Block Information

You can access the field values and selections you made in a block from the Edit menu or the diagram.

Accessing from the Diagram

To access block information from the Diagram Window:

1. Place your cursor over one block in the diagram.
2. Double-click your left mouse button. The selected block's dialog box appears.
3. Examine or change the parameters that have been set for the function block.

Accessing from the Edit menu

1. Select a block.
2. From the menu bar, select **Edit**. The Edit menu appears.
3. Select **Block Information**. The dialog box for that block appears.
4. Examine or change the parameters that have been set for the function block.

For additional information about specific blocks see the chapter that contains that type of block. The block chapters have been broken up according to the blocks general class of operation. The general classes of operation are:

- Control setting blocks
- Motion blocks
- I/O and event blocks
- Program control blocks
- Multitasking blocks
- Status blocks
- AxisLink blocks
- RIO blocks
- CNET blocks
- DH-485 blocks
- SLC blocks

- Calculation blocks
- Display and operator interface blocks
- Miscellaneous blocks

Chapter 6

Working with Diagrams

Diagrams are created in the Diagram Window using Commander's graphical elements. They are the graphical equivalent of scripted programs. By creating a diagram you build the instruction set for running your job.You can create original diagrams or copy and paste parts of existing diagrams into your new diagram.

Creating a Diagram

Diagrams can be created in two ways:

- Open a new Diagram Widow, add your elements, and save it.
- Open an existing diagram, modify it, and save it with a new name.

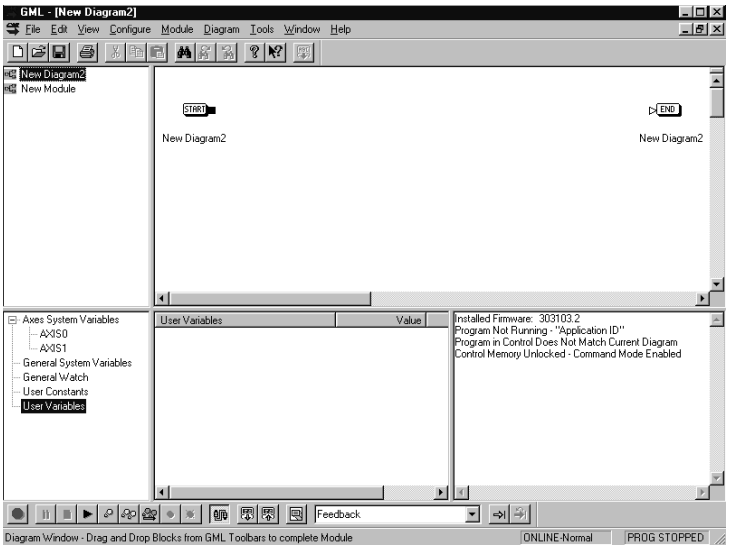
Building a New Diagram

Each time you open GML Commander, a new Diagram Window appears, ready for you to begin developing a new instruction set. You can also use the New Diagram option on the File menu.

To build a diagram:

1. Select **New Diagram** from the **File** option of the menu bar.

2. A new GML Commander diagram window similar to the following appears:



3. Rename the diagram using the Save As option.

4. Use the GML Commander function blocks and/or existing modules to build your diagram.

Use Preexisting Modules to Create Diagrams

In many cases an existing diagram, or part of an existing diagram, contains instructions very similar to the ones you need. You can recycle part of a diagram (usually a module) for use in your new diagram.

You can conveniently recycle commonly used configurations by using modules. Modules can be very complex and their creation can be time-consuming. The process for using preexisting modules, or parts of diagrams, involves copying and pasting.

We suggest that you create a library for your diagrams on your computer's hard drive or your organization's network to use as an information resource.

You can copy and paste modules from preexisting diagrams by using the standard Windows copy and paste features.

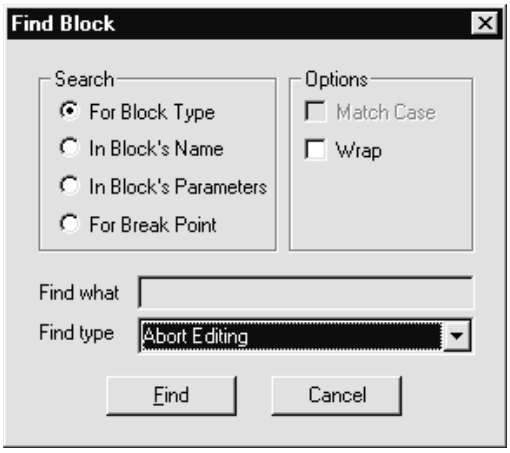
Important: When you copy blocks or modules from one diagram to another, check the parameters for the blocks and change them to meet the requirements of the new instruction set.

Finding a Specific Block or Parameter

The Find option on the Diagram menu works a little differently from the standard Windows Find function. It lets you search for elements in the current diagram using block specific terminology.

To search for elements:

1. Select **Find** from the **Edit** option on the menu bar
2. A dialog box similar to the following appears.



3. In the *Search* area, make entries in the following fields:

Field	Description
For Block Type	Select to activate the drop-down list of the *Find type* field that you use to search for a specific type of block.
In Block's Name	Select to activate the *Find what* field that you use to search for a block containing all or part of the name that you type.
In Block's Parameters	Select to activate the *Find what* field that you use to search for a block containing all or part of the expression value that you type.
For Break Point	Select to search for a break point within a diagram.

4. In the *Options* area, make entries in the following fields:

Field	Description
Match Case	Select this to search for text matching exactly what you type in the *Find what* field, including case.
Wrap	Select this to make the search automatically wrap to the beginning of the diagram when it reaches the end of the diagram.

5. Make entries in the following fields:

Field	Description
Find what	Type the text that you want to search for. If a block contains the text, the block appears in the Diagram Window. **Note:** This field is activated only if the *In Block's Name* or *In Block's Parameter* field is selected in the Search area.
Find type	Select the type of block that you want to search for from the drop-down list. **Note:** This field is activated only if the *For Block Type* field is selected in the Search area.

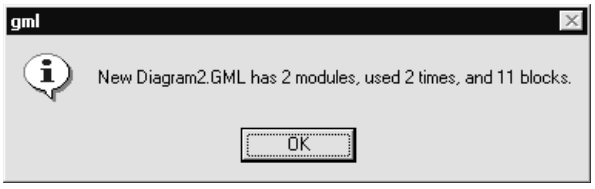
6. Select **Find**. The system searches for the information.

Displaying Diagram Information

GML Commander has a Diagram Info option to display module and block information for the current diagram. The message box states the number of discrete modules in the active diagram, the number of times modules are used in the diagram, and the number of discrete function blocks the diagram contains.

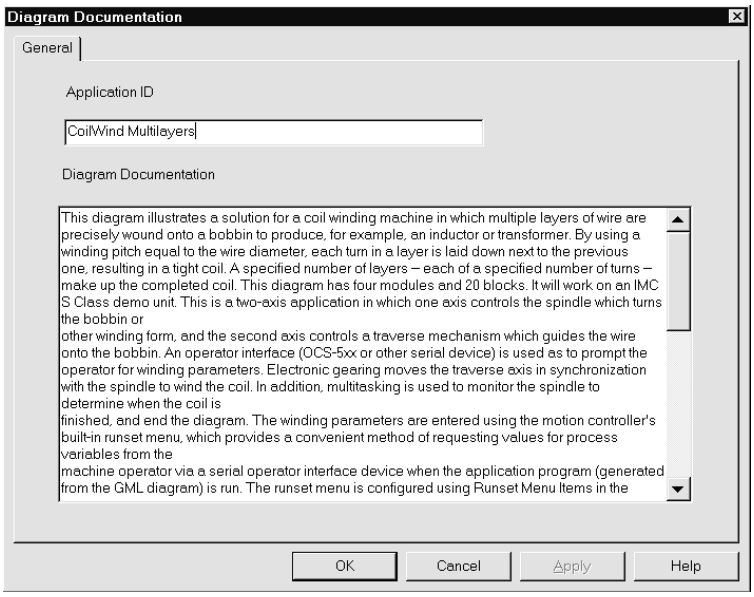
To display this information about the current diagram:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Diagram Info**. A dialog box similar to the following appears:



Documenting Your Diagram

Documentation provides a place to record information about a specific diagram. To document information about your diagram, select **Diagram Doc** from the Diagram menu option. The following screen displays

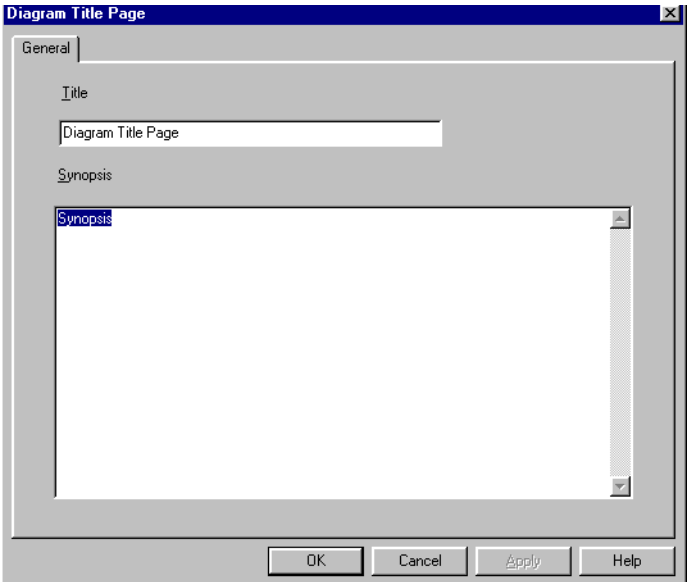


Type your information regarding the diagram and press OK. The information is saved and the dialog box disappears.

Diagram Title Page

The Diagram Title Page lets you enter the title of the diagram and provide a brief synopsis of the diagram ‘s function. To use, select Diagram Title Page from the Diagram menu option. The following screen displays

:



Enter the diagram title and a brief synopsis of the diagram in the areas provided.

Testing Diagrams

Use the Test Diagram option to test your diagram for valid translation into a program for the selected motion controller. This feature includes tests for valid block parameters and block connections. If GML Commander finds an error, a message box appears, describing the error and identifying its source.

To test your diagram, select **Test Diagram** from the **Diagram** menu option. If one or more errors is present, the first of the warning messages appears with the source of the error highlighted in the background. Select **OK**. If the error is a parameter within a block, GML Commander automatically opens the block and presents the dialog box containing the error

.

If there are:	Do this:
Errors	1. Correct the error(s). 2. Go to step 1.
No errors.	Your test is complete

Important: A successful test indicates that the diagram translates to a valid program, not that the motion controller operates your machine flawlessly.

Translating a Diagram into Program Script

Before you perform a download, you can translate the diagram to program script. We strongly suggest that only an experienced programmer should edit a program.

Important: Changes made to a program are not converted to changes in the corresponding diagram.

To convert your diagram into a program script: Select **Translate to Script** from the **Diagram** menu option. The program appears in a window similar to the following:

```
.D
.FA=""
.FR=
.FA="Application ID"

(B33=0):(B46=0):(B44=0):(B45=0):(B48=0):(B29=0):(B53=0)
(D{0}31=2):(B{0}9=0)
(D{0}48=6):(D{0}49=3)
(D{0}50=4):(D{0}51=0)
(D{0}29=6):(D{0}30=3):.FP{0}="inches":.FPS{0}="inches"
(D{1}31=2):(B{1}9=0)
(D{1}48=6):(D{1}49=3)
(D{1}50=4):(D{1}51=0)
(D{1}29=6):(D{1}30=3):.FP{1}="inches":.FPS{1}="inches"
(D{4}31=0)
.FX[0,0,0]
.FX[1,0,0]
.FX[2,0,0]
.FX[3,0,0]
.FX[4,0,0]
.FX[5,0,0]
.FX[6,0,0]
.FX[7,0,0]
.L0
.U
END HEADER

H{0}
```

Accessing the Online Manager

Select the Online Connection option on the Diagram menu to access the Online Manager.

For more information about the Online Manager, refer to the *Going Online* chapter.

Inserting a Breakpoint

Use the Breakpoint option on the Diagram menu to set or clear breakpoints. You can set and clear breakpoints using the Diagram menu or Online Toolbar but you must be online to set a breakpoint.

Breakpoints can only be set after the diagram is downloaded. The diagram must be downloaded every time a change is made to the diagram.

Select **Download Diagram** from the **Diagram** menu option. To display the Breakpoint Control dialog box, select the block where you want to set a breakpoint.

Select **Insert/Remove Breakpoint**. A black validation check mark appears next to the block with a breakpoint that is set.

Note: Select Clear All Breakpoints to clear breakpoints.

For more information about the breakpoint option, see the *Going Online* chapter.

Chapter 7

Working With Modules

A module is one block made up of several related blocks and/or modules. These blocks and modules together comprise either a larger function or related functions. Using modules allows you to hide detail and save space. You can duplicate, copy, move, connect, and encapsulate a module just like other blocks in the diagram.

Creating a Module

You can create a new module by doing any one of the following:

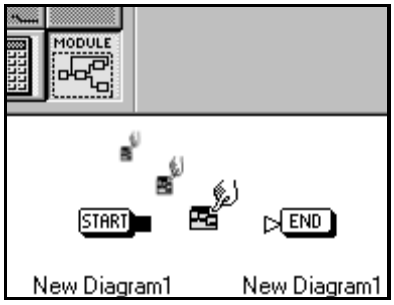
- Select the New Module block from the Main palette.
- Select the New Module block from the Diagram Explorer.
- Encapsulate several blocks.
- Copy, paste, and rename an existing module to use within the same diagram or from another diagram.
- Duplicate an existing module within a diagram as if it was another call of the original module.

Using the Main Palette

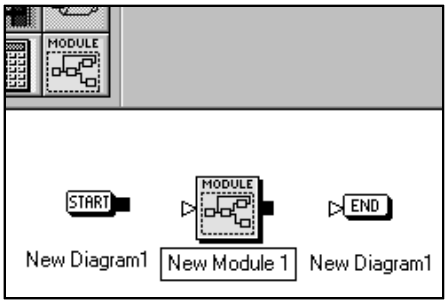
Use this method when you want to create a diagram by starting with a generalized idea. For example, each module might be a major program function, such as startup or setup. A new module is empty when you select it from the palette. You can assign an appropriate name and add blocks, block parameters, and connections later.

To create a new module:

1. From the Block palette, select the **New Module** block.



2. Drag the block to where you want to place it and click again. The block appears in the workspace.

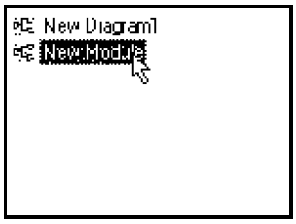


Using the Diagram Explorer

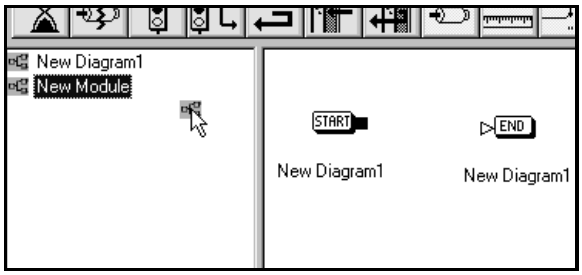
Use this method when you want to create a diagram by starting with a generalized idea. For example, each module might be a major program function, such as startup or setup. A new module is empty when you select it from the Diagram Explorer. You can assign an appropriate name and add blocks, block parameters, and connections later.

To create a new module:

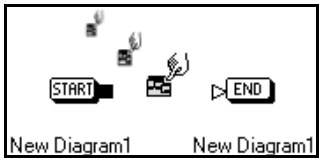
1. From the Diagram Explorer, select the **New Module** block.



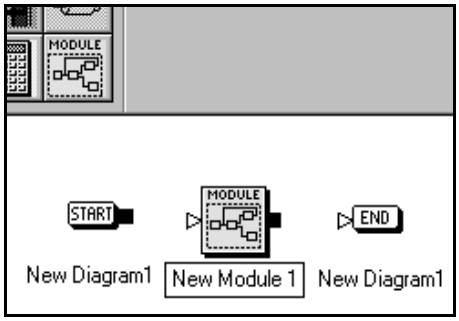
2. Hold down the left mouse button and drag the arrow cursor with the attached block to the Diagram Window. In the Diagram Window the arrow cursor becomes a finger cursor.



3. While holding down the left mouse button, place the finger cursor in the Diagram Window.



4. Release the left mouse button. The block appears in the workspace.



Important: A new module from the tree's node can also be dragged and dropped onto another tree's node within the Diagram Explorer.

Encapsulating Blocks

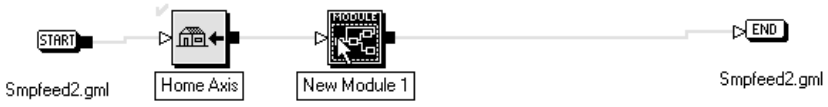
Create a new module to simplify the structure of your diagram. Use the encapsulate function to combine several blocks into a single module. You can easily reverse this process by unencapsulating the module. Refer to the *Unencapsulating a Module* section.

Important: Do not attempt to encapsulate a group of blocks that could result in multiple inputs or outputs. This results in lost connections.

To create one new module by encapsulating several blocks:

1. Select the blocks and connections that you want to encapsulate.
2. From the menu bar, select **Module**. The Module menu appears.

3. Select **Encapsulate**. The selected blocks are encapsulated into a module and given the default name New Module.



4. Change the default name to a name meaningful to you.

Duplicating an Existing Module

Once you have created a module that performs a function, you can duplicate the module. This is not a copy of the original, but another occurrence of the original. The duplicate has the same name and links as the original. Therefore, if you change the contents of any occurrence of the duplicated module, all of the duplicated are changed.

The primary use of a duplicate is to perform the same function more than once within the same diagram.

To duplicate an existing module:

1. Drag the module to the workspace.
2. From the menu bar, select **Edit**. The Edit menu appears.
3. Select **Duplicate Module**. A new module, with an identical name and the same contents, appears in the active diagram next to the original.
4. Position and connect the new module.

Recursive Duplicate Modules

Modules are said to be “Recursive” when a module contains a duplicate of itself. Using recursive duplicate modules can lead to an endlessly executing loop, unless the programmer includes carefully designed logic that provides an escape from the loop.

Important: The number of module calls is limited to the size of the application program stack inside the controller. The following are the depth limits for module calls:

IMC S Class Basic, Integrated Compact and 1394:	25 calls deep per task
IMC S Class 1394 Turbo:	40 calls deep per task

ATTENTION: Recursion can lead to stack overflow. If stack overflow occurs, the controller does not issue a stack overflow error. Controller behavior can become unpredictable. Communications failure may occur, resulting in the inability to issue direct commands from GML Commander to the controller. Unpredictable program execution may result. Motion may not stop.

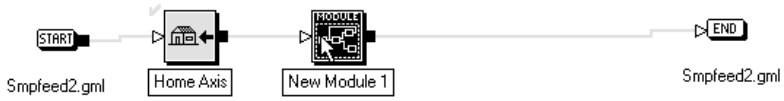
Unencapsulating the Module

If you've encapsulated several blocks into one module, you can reverse the process by unencapsulating the module.

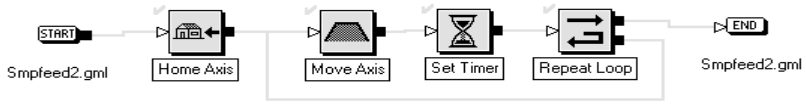
Important: If you unencapsulate a module that contains several blocks within a diagram level that already contains blocks, sorting the old blocks from the newly unencapsulated ones can be difficult.

To unencapsulate the contents of a module:

1. Select the module. The module is highlighted.



2. From the menu bar, select **Module**. The menu appears.
3. Select **Unencapsulate**. The module separates into its component parts.



Note: To view the blocks in the module without unencapsulating, refer to the *Viewing Module Details* section.

Viewing the Contents of a Module

It is usually easier to view and work with complex diagrams by encapsulating functionally distinct parts into separate modules. GML Commander lets you do this without actually changing the structure of the diagram.

To view the contents of a module, do one of the following:

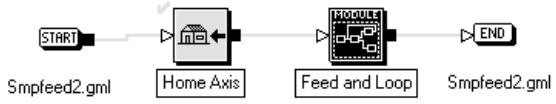
- Select a module in the Diagram Window and then use the Expand option to visually open the module and view the blocks and modules contained within.
- Double-click on the module block in the Diagram Window.

- Select a single module in the Diagram Explorer and view its contents in the Diagram Window.

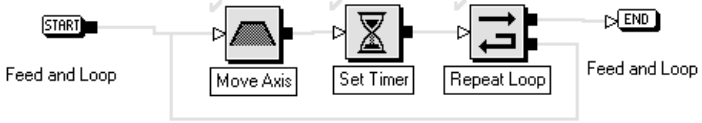
Viewing Module Contents Using Expand

To view the contents of a module while preserving the module:

1. In the Diagram Window, click once with the left mouse button to select a module block within a diagram. The module block is highlighted. In the following example, the Feed and Loop module block is highlighted



2. From the menu bar, select **Module**. The Module File menu appears.
3. Select **Expand**. The module block in the Diagram Window is replaced by the contents the module. In the following example, the contents of the Feed and Loop module appear (The name of the module appears under the Start and End blocks).



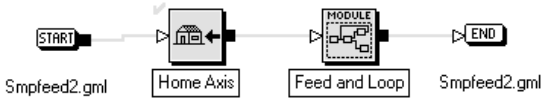
Collapsing Modules

There are several ways to collapse an expanded module in the Diagram Window, causing it to display the module block (in place of the blocks and connections within it):

- In the Tag Explorer tree diagram, select a module that is one or more levels above the expanded module.

- Select Collapse Module in the Module menu (or in the pop-up menu that opens by clicking the right mouse button with the pointer in the Diagram Window.
- Select Collapse Navigation then select the name of the module to display, from the pop-up menu that opens by clicking the right mouse button in the Diagram Window.

The original diagram appears in the Diagram Window.

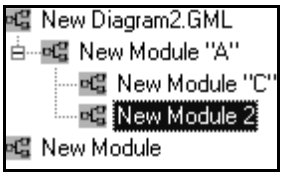


Viewing Module Contents Using the Module Block

To display the contents of a module in the Diagram Window, double-click on a module block in the Diagram Window. The module's details appear in the Diagram Window.

Viewing Module Contents In the Diagram Explorer

To display the contents of a module in the Diagram Window, select a module in the Diagram Explorer. The module's contents appear in the Diagram Window. An example of how the selected module appears in the Diagram Explorer is shown below:

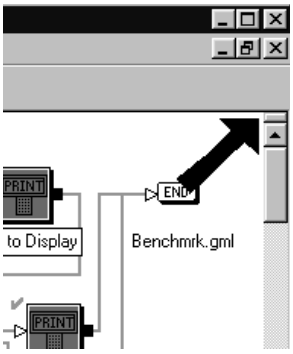


Splitting the Diagram Window

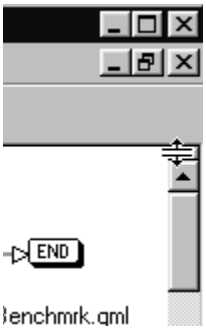
To see different areas of the same diagram, you can split the Diagram Window into two viewing areas, each with its own vertical scroll bar. This enables you to work on separate modules within the same diagram.

With a diagram displayed, split the Diagram Window by doing the following:

1. Locate the splitter bar in the top, right-hand corner of the Diagram Window. The arrow shows the splitter bar above the scroll-bar.

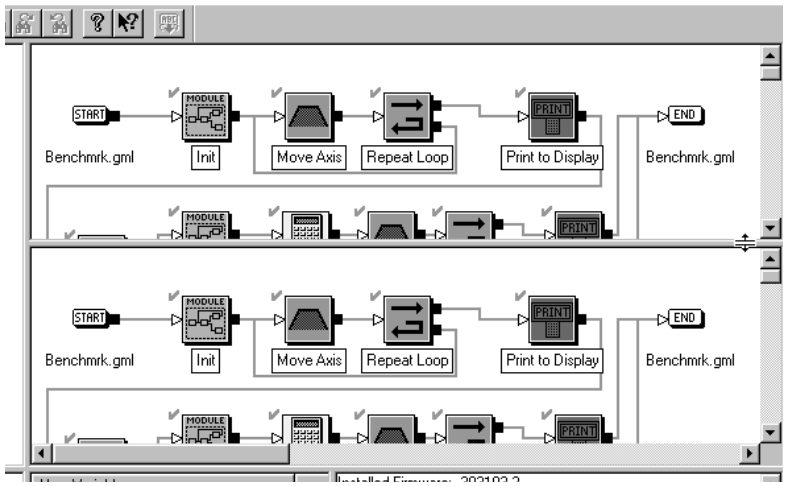


2. Move your cursor to touch the splitter bar. The cursor changes shape as shown below:

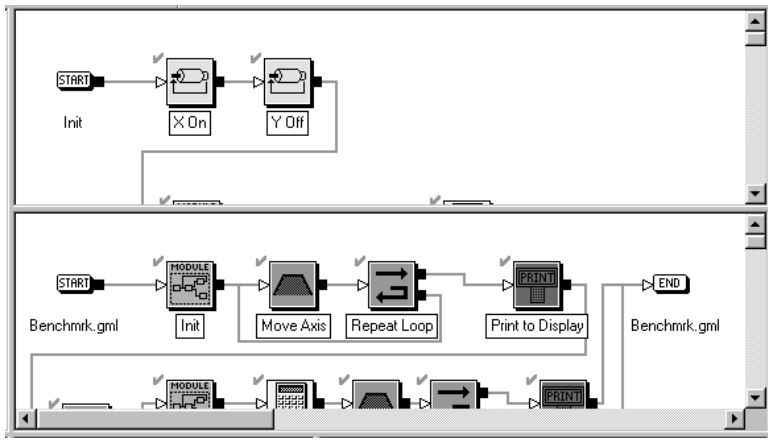


3. While holding down the left mouse button, pull the splitter down. The Diagram Window splits into two views.

Note: You can also split the Diagram Window by clicking in the Diagram Window with the left mouse button and selecting Split from the Window menu.



4. At this point you can, for example, double-click with the left mouse button on a module in one of the views. In the example below, the contents of the Init module was opened for viewing.



You can continue to change these two working views of the Diagram Window. This would enable you to view two widely separated areas of a large diagram. In addition, you can display different modules in either view by selecting them from the Diagram Explorer.

To change focus to the lower view, hold the Shift key down while clicking the left mouse button on a module node in the Diagram Explorer.

Displaying Module Information

To determine the number of blocks and immediate descending modules at any level:

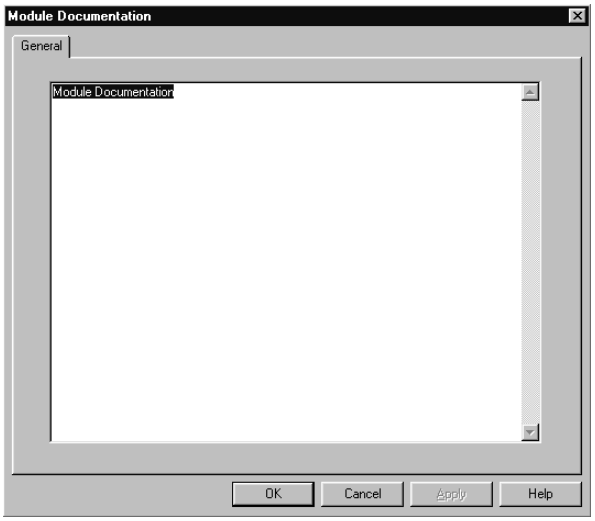
1. In the Diagram Explorer, select the desired diagram or module level.
2. In the Diagram Window, select the module that you want details about. The module is highlighted.
3. From the menu bar, select **Module**. The Module File menu appears.
4. Select **Module Info**. The dialog box similar to the following appears, showing the number of blocks and the number of modules in the selected module level.

Documenting a Module

You can enter information about a module that can be helpful to you and future users. The remarks you enter stay with the module and are not diagram specific. The documentation can be accessed for nay diagram that uses this module.

To document a module that appears in the Diagram Window:

1. In the Diagram Explorer, select the diagram or module level that contains the module that you want.
2. Select the module that you want to document. The module is highlighted in the Diagram Explorer.
3. From the menu bar, select **Module**. The Module menu appears.
4. Select **Module Doc**. The Module Documentation dialog box appears. The name of the active module displays above the documentation input box.



5. Type new information or change the existing information.
6. Select **OK**. The information is saved and the Diagram Window reappears.

Chapter **8**

Working with Program Scripts

A program script is a text-based program file in iCode format. The controller requires programs in iCODE. GML Commander diagrams are not in iCODE format and it must be translated to iCODE for the controller to understand your commands. The Script Editor in GML Commander allows you to view and modify iCODE that was either translated from a diagram or uploaded from a controller.

Important: We recommend that you use GML Commander's graphical interface (create a diagram) to program your controller and not script. When you edit a program file using the Script Editor, the changes you make are not made to the diagram.

Translating a Diagram to Script

When you download your diagram to the controller, GML Commander translates your diagram to iCODE, the native language of the controller.

You can translate a diagram to script before downloading it, but changes to the script cannot be converted back to the graphical diagram. If changes to the script are permanent, the changed script, and not the original GML Commander diagram, becomes the program for the download.

This option automatically includes a testing function. The translation process is very similar to the testing process. When a problem is identified, a dialog box informs you of the problem and its location. It automatically goes to that block and opens the block's dialog box so you have ready access to fix the problem.

To translate a graphical GML Commander diagram to a native language script it must be the active diagram in the window. Select **Translate to Script** from the Diagram menu option to begin translation of the active diagram. As GML Commander translates the diagram into a script, a dialog box informs you of the translation progress.

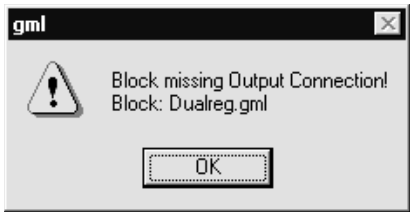
Successful Translation to Script

A successful translation of a diagram to a script causes the script workspace to replace the diagram workspace. The script displays, similar to the example below:

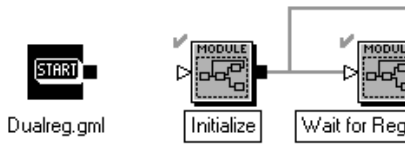
Important: Once you convert a diagram to script, the procedure cannot be reversed. The script text is the actual language used for the motion controller program.

Handling Unsuccessful Translations

During the translation, the GML Commander translator checks for syntactical errors. If it finds errors and the translation is not successful, a dialog box appears, identifying the problem and its location. As shown in the following exhibit.



To correct a diagram error, select OK in the GML dialog box. Commander automatically takes you to the section of the diagram where the error occurred.



Make the appropriate corrections to the diagram and when complete run the Translate to Script procedure from the Diagram menu options.

Editing a Script

The Script Editor can be used to edit and view the script text files. You can create a new script by translating a GML Commander diagram to script or by uploading the program from your controller.

The Script Editor provides standard cut, copy, paste, delete, and find/replace functions. They work in the same manner as these functions work in all Windows programs.

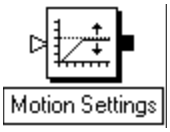
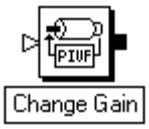
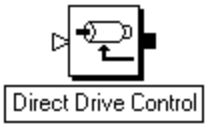
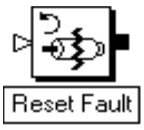
Important: Changes to the script cannot be converted back to a graphical diagram.

Use the Open Script option from the File menu option to open a saved script. You can then use all or portions of the script to build your new program. When the existing script opens, you can use all or part of the script to meet your new needs. However, we recommend that you create new programs using the diagram procedure rather than the script procedure.

Control Setting Blocks

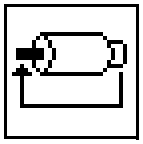
Control Setting blocks directly control some aspect of the motion controller.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Turn feedback on or off.
	Set the motion controller's: • output limit for a physical servo axis • motion profile for a servo axis • maximum speed (velocity) value for a servo axis • maximum acceleration value for a servo axis
	Change any physical servo axis' working values for: • proportional gain • integral gain • velocity gain • feedforward gain • deadband compensation
	Set an axis' servo output to the commanded voltage or current.
	Clear: • all axis faults on a selected physical or virtual axis • a particular fault on a selected physical or virtual axis • 1394 system faults

Use this block:	To:
	Set an axis' actual or command position to a commanded absolute or relative position.
	Show, adjust or read the current working or power-up value of a selected data parameter or data bit

Feedback



The Feedback block directly and immediately affects the appropriate drive enable output and the feedback loop processing on any servo axis.

The Feedback block resides on the Main Palette.

You can use the Feedback block anywhere in a GML Commander diagram. However, do not use the Feedback block while the axis is moving, because this causes an abrupt, uncontrolled stop.

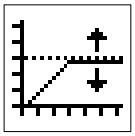
If the motion controller's feedback state is ON:

- Feedback_status = 1
- Axis_fault = 0, and Axis_status ≤ 5 if no faults are active on the axis
- Global_fault = 0 if no other faults are active on any axis

When feedback is OFF:

- Feedback_status = 0
- Axis_fault = 0, and Axis_status = 6 if no faults are active on the axis
- Global_fault = 0 if no other faults are active on any axis

Motion Settings



The Motion Settings block allows you to set the motion controller's:

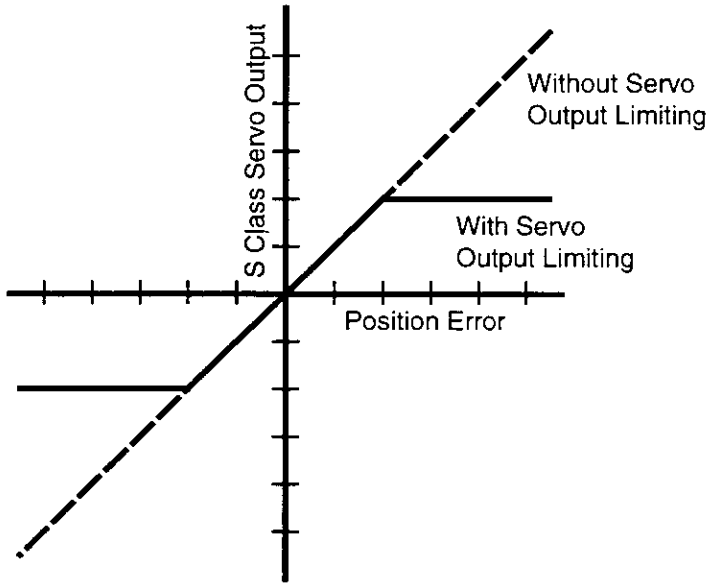
- Output limit for a physical servo axis
- Motion profile for a servo axis
- Maximum speed (or velocity) value for a servo axis
- Maximum acceleration value for a servo axis
- Maximum deceleration value for a servo axis

The Motion Settings block resides on the Main Palette

Important: The power-up values, set in the Configure Axis Use dialog box, are unchanged by the execution of the Motion Settings function.

Loop Error

The figure below shows the motion controller's servo output as a function of loop error, both with and without servo output limiting.



When the servo output is being clamped at the servo output limit

- Output_limit_status = 1
- Axis_status = 5 if no faults are active

When the servo output of an axis is not being clamped at the servo output limit

- Output_limit_status = 0
- Axis_status ≤ 4 if no faults are active on the axis

Unit Settings

If the motion controller is a 1394 or a Compact, you can specify the servo output in volts, percentage of full scale, or mA as required by your application. The Compact (IMC-S/23x models) provides a user-configurable servo output that can be set for ±10V (standard) or ±150 mA output (for use with hydraulic servo and proportional valves).

Integrated (IMC-S/21x models) and Basic (IMC-S/20x models) motion controllers provide a ±10V servo output for each physical axis. If you are using one of these motion controllers, you can specify the servo output either directly in volts or as a percentage of full scale as required.

Using the Servo Output Limit

You can use the servo output limit as a software current or torque limit (clamp) if you are using a servo amplifier in torque (current) mode. The maximum command the motion controller ever sends to the servo amplifier, is equal to the specified servo output limit. For example, if your amplifier is capable of 30 Amps of current for a 10 Volt input, setting the servo output limit to 5 Volts or 50% limits the maximum motor current to 15 Amps.

You can also use the servo output limit if your servo amplifier cannot accept the full ±10 Volt range of the motion controller's servo output. For example, if your servo amplifier can only accept command signals up to ±7.5 Volts, set the servo output limit to 7.5 Volts or 75%.

Setting the Motion Profile

You can also use the Motion Settings block to set the type of motion profile used with all subsequent Move or Jog Axis blocks for the selected axis. It does not change the power-up profiles for the axis set in the motion controller's application setup menu, but merely changes the current working profile selection.

The motion controllers provide trapezoidal (linear acceleration and deceleration), S curve (controlled jerk), and parabolic velocity profiles. A guide to the effects of these three motion profiles on various application requirements follows:

Velocity Profile	Acc/Dec Time	Motor Stress	Mechanical Stress
Trapezoidal	Fastest	Worst	Worst
S-Curve	2X slower	Better	Best
Parabolic	2X slower	Best	Better

Note: For more information on motion profiles, see the "Defining Position" section of the "Configuring Axis Use" chapter in this manual.

Setting the Maximum Speed

You can use the Motion Settings block to set the 100% velocity value for all subsequent Move or Jog Axis blocks for the selected axis. This does not change the power-up maximum velocity value for the axis set in the motion controller's servo setup menu, but changes only the current working value.

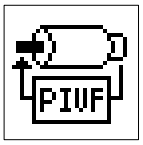
Setting the Maximum Acceleration

You can use the Motion Settings block to set the 100% acceleration value used with all subsequent Move or Jog Axis blocks for the selected servo axis. This does not change the power-up maximum acceleration value for the axis set in the motion controller's servo setup menu, but changes only the current working value.

Setting the Maximum Deceleration

You can use the Motion Settings block to set the 100% deceleration value for all subsequent Move or Jog Axis blocks, for the selected physical or imaginary axis. This does not change the power-up maximum deceleration value for the axis set in the motion controller's servo setup menu, but changes only the current working value.

Change Gain



The Change Gain block directly changes the working values of any physical servo axis':

- Proportional gain
- Integral gain
- Velocity gain
- Feedforward gain
- Deadband compensation

The Change Gain block resides on the Advanced Motion Palette.

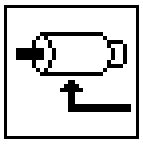
You can use the Change Gain block to change the working servo gains at any time. This does not effect the power-up values, entered using the motion controller's servo setup menu and stored in non-volatile application memory.

ATTENTION: Changing gains can cause an axis to become unstable. Refer to the Online Help for Gains for more information.

Gain Values

The measurement unit for each type of gain depends upon the motion controller you are using. Refer to you motion controller's documentation to confirm the measurement unit for each type of gain.

Direct Drive Control



The Direct Drive Control block directly sets the servo output of the selected physical axis to the commanded voltage, current, or percent of maximum. The maximum settings are either 10 Volts (V) or 150 milliAmperes (mA).

The Direct Drive Control block resides on the Advanced Motion Palette.

Units

For 1394 or Compact motion controllers, you can specify the servo output in volts, percentage of full scale, or mA as required in your application. For IMC-S/23x model Compact controllers, you can use a configurable servo output that can be set for ±10V (standard) or ±150 mA output (for use with hydraulic servo and proportional valves).

Integrated (IMC-S/21x models) and Basic (IMC-S/20x models) motion controllers provide a ±10V servo output for each physical axis. For these motion controllers, you can specify the servo output either directly in volts or as a percentage of full scale, as required in your application.

Using the Direct Drive Control Block

ATTENTION: The Direct Drive Control block assumes positive (non-inverted) servo output polarity for proper operation. If the servo output polarity bit for the axis is set (B11 = 1 or Enabled), the polarity of the output voltage is opposite that entered in the Direct Drive Control block. See the *Control Settings* block, particularly the *Show Control Setting* type, section in this chapter for more on data bits.

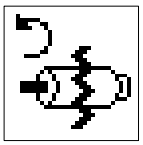
Use the Direct Drive Control block only on a physical master only axis, or a servo axis when feedback is off (with feedback on, it has no effect). The Direct Drive Control block automatically enables the selected axis by activating the appropriate drive enable output before setting the servo output to the commanded value. The 16-bit DAC on the motion controller limits the effective resolution of the Direct Drive Control block to 305 µV or 0.003%. In addition, the servo output limit (set by the Set Output Limit function of the Motion Settings block) ultimately limits the actual servo output voltage when using this block.

To deactivate the selected axis drive enable output, use a Feedback block (selecting the Feedback Off Type). This zeros the servo output and disables the drive enable output.

The Direct Drive Control block is most commonly used:

- to provide an independent programmable analog output
- to provide a speed reference for a non-servo drive
- for testing a servo drive.

Reset Fault



Use the Reset Fault block to clear
- All axis faults on a selected physical or virtual axis
- A selected fault status on a selected physical or virtual axis
- 1394 system faults

The Reset Fault function block resides on the Main Palette.

The Reset Fault block only removes the fault status. It does not perform any other recovery (for example, re-enabling feedback).

ATTENTION: Be sure to find and correct the cause of the fault before clearing the fault status. Otherwise, the fault recurs on program start-up.

Selecting Reset 1394 clears all of the following 1394 system faults:

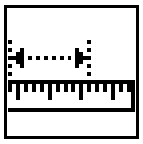
- Axis_bus_loss_fault_1394
- Axis_Drive_over_temp_fault_1394
- Axis_it_fault_1394
- Axis_module_hard_fault_1394
- Axis_Motor_over_temp_fault_1394
- Axis_power_fault_1394
- Drive_hard_fault_1394
- DSP_feedback_fault_1394
- System_bus_over_voltage_flt_1394
- System_bus_undr_voltage_flt_1394
- System_control_power_fault_1394
- System_ground_fault_1394
- System_over_temp_fault_1394
- System_phase_loss_fault_1394
- System_serial_ring_fault_1394
- System_smrt_pwr_i_limit_fault_1394
- System_smrt_pwr_pre_charge_fault_1394
- System_smrt_pwr_shunt_timeout_fault_1394

Selecting Reset 1394 clears the latched fault only if the fault itself is not active. In addition, this selection causes the controller to try to re-initialize the drive module(s) interface with their configured parameters.

Selecting Reset 1394 does not clear axis faults that are cleared by the Axis Fault or All Faults selections.

See the *Fault Variable* chapter in this manual for more information on the specific faults.

Redefine Position



The Redefine Position block sets the actual or command position of the selected axis to the commanded absolute or relative position. This block causes no motion—it merely redefines the current axis position.

The Redefine Position block resides on the Main Palette.

You can use the Redefine Position block when the axis is moving, and when the axis is at rest. You use the Redefine Position block to redefine position on the fly, and for certain registration, slip compensation, and re-calibration applications.

Absolute Mode

When you select Absolute from the Mode menu, the new position value specifies the new absolute position of the axis. No motion occurs—the current axis position (actual or command) is merely redefined, and becomes the commanded new position.

If your motion controller uses software overtravel limits, the new position must be a value between the maximum positive and maximum negative travel parameter values. Otherwise, executing this block generates a software overtravel fault.

ATTENTION: When you redefine an absolute position while the axis is moving, you can get unexpected results for certain types of moves.

Absolute and Relative selections have the same effect when the axis is not moving. When the axis is moving, however, absolute mode introduces a position error equal to the motion of the axis during the time it takes to execute the Redefine Position block and assign the new position. Relative mode does not introduce this error and guarantees an exact correction independent of axis speed or position.

Relative Mode

When you select Relative from the Mode menu, the new position value offsets the current position of the axis. No motion occurs—the current axis position (actual or command) is simply redefined, and becomes the sum of the current position unit value plus the specified new position unit value.

In relative mode, this block redefines axis position in such a way that no position errors are introduced if the axis is moving. This block is particularly useful for unwinding axis position under program control, rather than using the built-in rotary axis feature.

Absolute and Relative mode Redefine Position blocks have the same effect when the axis is not moving. When the axis is moving, however, absolute mode introduces a position error equal to the motion of the axis during the time it takes to execute the Redefine Position block and assign the new position. Relative mode does not introduce this error and guarantees an exact correction independent of axis speed or position.

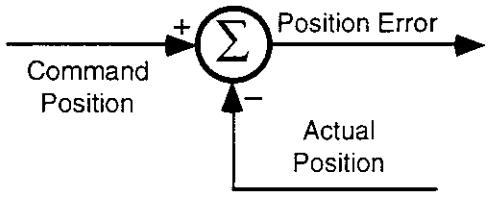
Actual Position

When you select Actual from the Position menu, the motion controller directly applies the new position value to the actual position of the physical or virtual axis. Because the imaginary axis does not have an actual position, you can select only Command from the Position menu for the imaginary axis. The motion controller also adjusts the command position of servo axes to reflect the new actual position and to preserve any position error that exists. This ensures that there is no unexpected axis motion when the positions are redefined. See *Command Position* for more information on command position, actual position, and position error.

Command Position

When you select Command from the Position menu, the motion controller directly applies the new position value to the command position of the servo or imaginary axis. Because master only axes do not have a command position, you can select only Actual from the position menu for master only axes. The motion controller also adjusts the actual position of servo axes to reflect the new command position, and to preserve any position error that exists. This ensures that there is no unexpected axis motion when the positions are redefined.

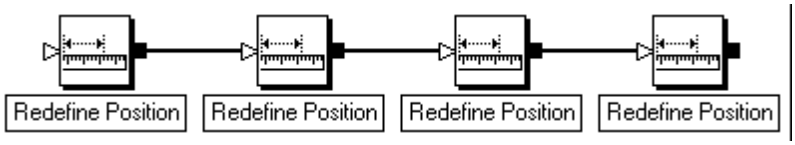
Command position is the desired or commanded position of a servo or the imaginary axis, as generated by any previous motion blocks. Actual position is the current position of a physical or virtual axis as measured by the encoder or other feedback device. Position error is the difference between these two and is used to drive the motor to make the actual position equal to the command position. The figure below shows the relationship of these three positions.



See your motion controller's *Installation and Setup* manual for information about the nested digital servo loop used in the motion controllers.

Synchronizing Redefine Position Blocks

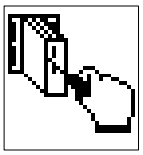
When you select Synchronize with next block, the action of this Redefine Position block occurs simultaneously with the following Redefine Position block. This lets you redefine the position of multiple axes simultaneously, as shown below.



When you select Synchronize with next block, the block immediately following the Redefine Position block must be another Redefine Position block. GML Commander does not permit a connection to be made to any other kind of function block.

In the example above, select Synchronize with next block in the first three Redefine Position blocks—but not in the last block—to simultaneously redefine all four axes' positions.

Control Settings



Use the Control Settings block to:

- Send the current working or power-up value of the selected data parameter or data bit (Show Type) to the operator interface port
- Change the current working or power-up value of the selected data parameter or data bit (Adjust Type)
- Store the current value of the selected data parameter or data bit (Read Type) in a selected user variable

The Control Settings block resides on the Main Palette.

Power-Up Values

When you select Power-up from the Value menu, the motion controller shows, adjust or read the current power-up value (or state) of the data parameter or data bit selected from the scrolling list, depending upon the command Type selected. (See *Data Parameters* and *Data Bits in Appendix A* for more information.) The power-up values are the values entered in the Configure Control Options dialog box, and in the Configure Axis Use dialog box.

Whenever you power-up the motion controller—or whenever you press the Reset button, the working values are initialized (set equal) to the power-up values. At this moment, immediately after power-up, the power-up and working values of each data parameter or data bit are identical. Each time you run the application program in the motion controller, the motion controller again initializes the working values (i.e., sets them equal to the power-up values). However, the program itself may modify the working values, thereby making the working values of each data parameter or data bit different from the power-up values.

ATTENTION: Do not attempt to change the power-up values, of data parameters or data bits, when the motion controller memory is locked.

The power-up values are locked, and can not be changed, when you do either of the following:

- lock the motion controller memory using the front panel keyswitch (1394-SJTxx, IMC-S/21x, and IMC-S/23x models).
- remove the memory unlock jumper (IMC-S/20x models).

Attempting to change the power-up values with an Adjust type Control Settings block within a GML Commander diagram results in an Attempt to Access Locked Memory runtime fault (Runtime_fault = 22) when the program is executed. See *Runtime Fault* in the *System Variables* chapter of this manual for more information on runtime faults.

Working Values

When you select Working from the Value menu, the motion controller shows, adjust or read the current working value (or state) of the data parameter or data bit selected from the scrolling list, depending upon the command type selected. (See *Data Parameters* and *Data Bits in Appendix A*) The working values are the values used by the motion controller while it is running the application program.

Whenever you power-up the motion controller—or whenever you press the Reset button, the working values are initialized (i.e., set equal to the power-up values). At this moment, immediately after power-up, the power-up and working values of each data parameter or data bit are identical. Each time you run the application program in the motion controller, the motion controller again initializes the working values (i.e. sets them equal to the power-up values). However, the program itself may modify the working values, thereby making the working values, of each data parameter or data bit, different from the power-up values. In addition, you can change the working values either directly using an Adjust Type Control Setting block, or indirectly through the action of other blocks.

Tag Explorer

Select either Data Bits or Data Parameters.

Tag Window

Select the Data Bit or Data Parameter that you want to read, show or adjust.

Data Parameters

When you select Data Parameter, the available data parameters appear in a scroll-down list. A list of these data parameters appear in the Data Parameter table located in *Appendix A* of this manual.

ATTENTION: Do not enter values that do not fit the selected Value Format. Unpredictable operation may occur.

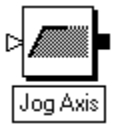
Data Bits

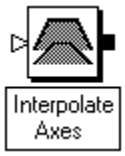
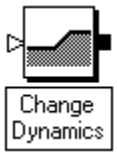
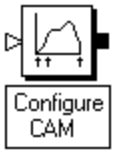
When you select Data Bits in the lower-left window, the available data bits appear in a scroll-down list in the lower-right window. These data bits are displayed in the Data Bits table located in *Appendix A* of this manual.

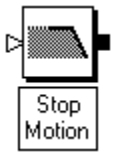
Motion Blocks

Motion blocks include all blocks that cause or directly affect motion of the physical axes. GML Commander provides blocks for homing, moving, jogging, electronic gearing, multi-axis interpolation, and electronic cams.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

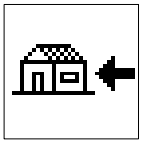
Use this block:	To:
	Home an axis and define that axis' actual position.
	• Move a physical or imaginary axis to a specified absolute position, or by a specified incremental distance. • Perform other special types of moves.
	Jog (that is, continuously move) a physical or imaginary axis in a specified direction, at a specified speed, using specified acceleration and deceleration values
	Electronically gear two axes at a set ratio.

Use this block:	To:
	Move a group of two or more physical or imaginary axes either to an absolute position on, or by a specified incremental distance, along a linear, circular or helical path.
	Change the: • Speed of trapezoidal profile moves • Speed, acceleration, and deceleration of trapezoidal profile jogs.
	Let the motion controller use an analog input to generate a scaled position offset, and add that offset to the command position of the specified servo axis.
	Set up the necessary conditions for executing a time-lock or position-lock cam, or to set up a pending position-lock cam that blends one position-lock cam profile into another.
	Executes a time-lock cam profile created by a previous Configure Cam block. Time-lock cams allow the execution of motion profiles other than the built-in trapezoidal, S-curve, or parabolic profiles
	Execute a position-lock cam profile created by a previous Configure Cam block. Position-lock cams provide the capability of implementing non-linear electronic gearing relationships between two axes.
	Stop a position-lock cam block from executing. The axis stops immediately with no deceleration.

Use this block:	To:
	Stop the electronic gearing motion of a selected Axis. If gearing is the only motion in progress on the axis, the axis stops immediately with no deceleration.
	Stop all motion, or any selected type of motion, on the selected physical or imaginary axis or interpolator. Except when used to perform an Emergency Stop, this block does not disable feedback.

Motion blocks provide access to all of the high-level motion functions in the motion controllers. See the *Installation and Setup* manual of your motion controller for an introduction to the high level motion functions.

Home Axis



The Home Axis performs a homing operation, as selected in the Procedure menu, and re-defines the actual position of the selected axis.

The Home Axis block resides on the Main Palette.

You cannot home an imaginary axis. Instead, use a Redefine Position block to change the actual position of an imaginary axis.

A virtual axis can be homed only passively. Note that, in the Configure Axis Use dialog box, there is no Homing tab. This is because a virtual axis is configured for passive homing by default. Therefore, it doesn't matter whether you select Configured or Passive in the Home Axis block. In either case, passive homing results.

The Homing_status variable is 1 (true) while the homing is in progress. When active homing is complete (see *Configured Homing*, below) the axis is at the home position and Homing_status is 0 (false). See *the System Variables* chapter in this manual for more information on the Homing_status variable.

Configured Homing

When you select Configured from the Procedure menu, the motion controller homes a physical axis according to the power-up homing configuration settings, selected in the Homing page of the Configure Axis Use dialog box. Depending upon these settings, the Configured selection in the Home function block causes one of the following homing procedures:

- Active
- Passive
- Absolute
- Absolute Serial
- Absolute MV

See the *Setup* section of the *Installation and Setup* manual for your motion controller for information on setting up the power-up homing configuration. Also, see *Online Help* for the Configure Axis Use Homing dialog box for more information.

Passive Homing

When you select Passive from the Procedure menu, the Home Axis block re-defines the current actual position of a physical or virtual axis by setting it equal to the Home position, upon the next occurrence of the encoder marker. The Home position is set in the Homing page of the Configure Axis Use dialog box. Passive homing is most commonly used to set the Master Only axes (physical and virtual) to their markers. Passive homing does not command any axis motion.

The Homing_status variable is 1 (true) while the homing is in progress. When the marker is detected, the motion controller assigns the home position to the exact location of the marker, and Homing_status is 0 (false). See the *System Variables* chapter in this manual for more information on the Homing_status variable. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on the home position.

After initiating passive homing, the axis must be moved past the encoder marker for the homing sequence to complete properly. For closed-loop servo axes, this can be accomplished with a Move Axis or Jog Axis block. For physical master only axes, motion cannot be commanded directly by the motion controller, and must be accomplished via other means. For virtual axes, the motion must be commanded from the associated physical axis on the appropriate motion controller.

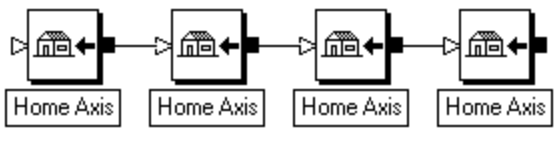
Wait for Completion

When you select Wait for Completion, the motion controller pauses the program until the chosen homing sequence is completed. When the homing sequence completes, the program continues with the next block. If other tasks are executing (multitasking), the motion controller halts the task that contains this block, but the other tasks continue to execute. In this way, a Wait for Completion selection in one task does not halt execution of any other tasks or hang the task dispatcher.

When Wait for Completion is not selected, the Home Axis block initiates the specified homing sequence and the program continues with the next block.

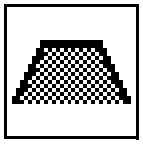
Synchronized Homing

When you select Synchronize with next Home Axis, the current Home Axis block executes simultaneously with the following Home Axis block. This allows homing multiple axes simultaneously, as shown below.



When you select Synchronize with next Home Axis, the block immediately following the Home Axis block must be another Home Axis block. Otherwise, GML Commander does not permit you to connect the two blocks. In the example above, Synchronize with next Home Axis should be selected in the first three Home Axis blocks—but not in the last block—to home all four axes simultaneously.

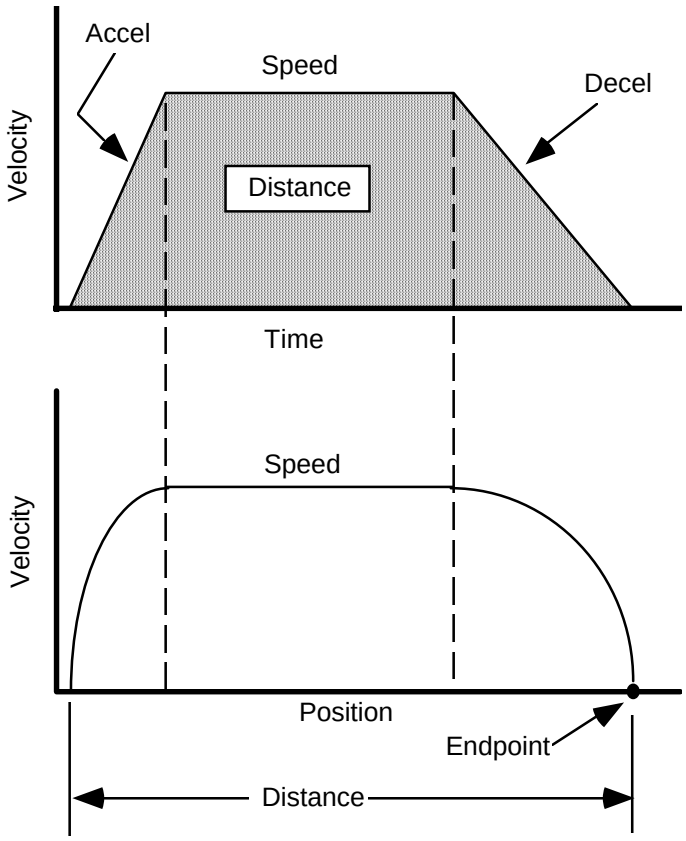
Move Axis



The Move Axis block moves a servo axis to a specified absolute position, or by a specified incremental distance, at a specified speed using a specified acceleration and deceleration. In addition to these absolute and incremental moves, the Move Axis block can also generate many other special types of moves.

The Move Axis block resides on the Main Palette.

The figure below shows the general form of a trapezoidal move starting with the axis at standstill.



While a move is in progress:

- Move_status = 1 (true)
- Axis_status = 3 (Moving)
- Lock_status = 0 (Unlocked)

When the move is done:

- Move_status = 0 (false)
- Axis_status = 1 (unlocked)

For physical axes, when the move is done and the position error is less than the position lock tolerance:

- Axis_status = 0 (Locked)
- Lock_status = 1 (Locked)

See the *Axis Locked and Axis Done Conditions* chapter in this manual for more information. See the *System Variables,* chapter in this manual for information on variables.

Absolute Moves

When you select Absolute in the *Move* field, the axis moves to the specified position at the specified speed, using the specified acceleration and deceleration.

Changing the Endpoint of Absolute Moves

You can change the endpoint of an absolute move while the move is in progress using another Move Axis block with a new absolute position. You can do this with any of the three velocity profiles (trapezoidal, S-curve or parabolic) except when the axis is accelerating or decelerating. You cannot use S-curve or parabolic profiles.

You can also change the endpoint of an absolute move using an incremental Move Axis block while the absolute move is in progress. In this case, the final destination of the axis is the original absolute position, plus the new incremental distance (see *Incremental Moves* in this chapter).

In all cases, the axis moves smoothly to the new endpoint without stopping at the old endpoint—including any required change of direction.

You can change the move speed of a trapezoidal absolute move in progress, and change the endpoint, by specifying a new speed in addition to the new position. If new acceleration and deceleration values are specified, they take effect only if, and when the axis reverses direction.

Absolute Moves on Rotary Axes

When an axis is configured for rotary operation, the motion controller handles absolute moves the same as with linear axes, except that when the axis position exceeds the unwind parameter, axis position is unwound. In this way, axis position never exceeds the unwind value, and never falls below zero.

The specified position is interpreted trigonometrically and can be positive or negative and can exceed the unwind value. Negative position values are equivalent to their corresponding positive values (i.e. –90° is the same as +270° etc.) and are useful when rotating the axis through zero. When the position exceeds the unwind value, the axis moves through more than one revolution before stopping at an absolute position.

Incremental Moves

When you select Incremental from the *Move* field, the axis moves the specified distance at the specified speed, using the specified acceleration and deceleration.

Changing the Move Distance

You can change the final destination of an incremental move while the move is in progress (but not while the axis is accelerating or decelerating with S curve or parabolic profiles) using another Move Axis block with an additional incremental distance. The total distance moved by the Move Axis blocks is the sum of the two distances.

You can also change the final destination of an incremental move while a move is in progress, using another Move Axis block with a new absolute position. In this case, the axis goes directly to the specified absolute position without completing the incremental move. This action is different from the case where an incremental Move Axis block follows an absolute Move Axis block.

Incremental Moves with Gearing

You can use a Gear Axes block while an incremental move is in progress on the slave axis. The gearing motions are superimposed on the move profile. Conversely, you can use an incremental move block while gearing is enabled to cause a similar effect. This allows you to accomplish many complex move profiles and sophisticated synchronization. Superimposing an incremental move on top of electronic gearing is particularly useful to accomplish phase advance/retard control.

Incremental Moves on Rotary Axes

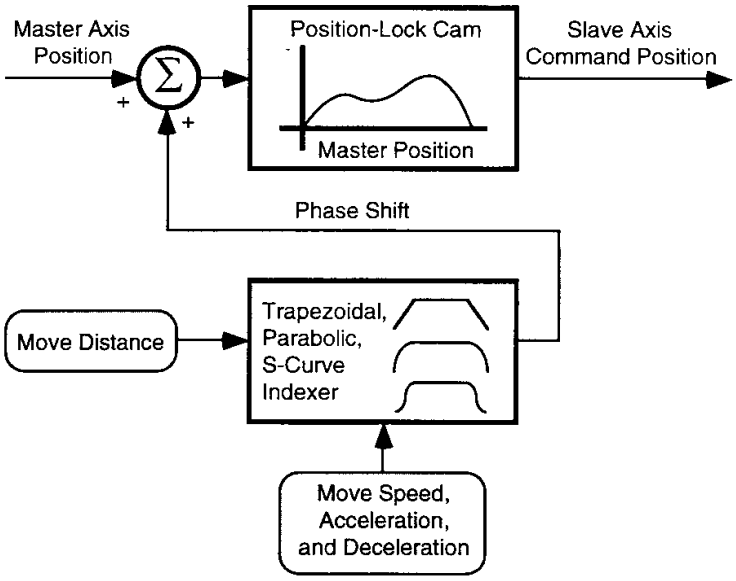
When an axis is configured for rotary operation, the motion controller handles incremental moves in the same way as with linear axes, except that when the axis position exceeds the unwind parameter, the axis position is unwound. In this way, axis position is never greater than the unwind value, and never less than zero.

The specified distance is interpreted trigonometrically and may be positive or negative and can be greater than the unwind value. When the distance is greater than the unwind value, the axis moves through more than one revolution before stopping.

Phase Shift Moves

You can use a Phase Shift move to shift a position-lock cam profile relative to its master axis position, thereby shifting the timing of the profile. In this way, the Phase Shift Move you can change the synchronization between the slave and master axes of a position-lock cam on the fly. This is analogous to phase shifting a mechanical cam by rotating it on its shaft. Phase Shift moves should only be used on the slave axis of a position-lock cam while the position-lock cam is executing. See the *Position Lock Cam* section in this manual for more information.

When you select Phase Shift, the currently executing position-lock cam profile shifts, relative to the position of its master axis, the specified distance at the specified speed using the specified acceleration and deceleration settings, as shown in the following diagram.

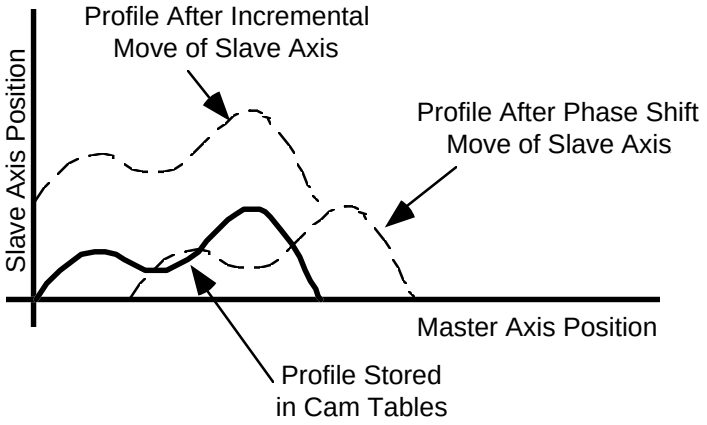


This motion can be thought of as smoothly shifting the master reference position of the cam by the specified distance, thus shifting the master axis positions relative to the slave axis positions.

ATTENTION: Although you command a phase shift move along the slave axis, always set the distance value in the position units of the corresponding master axis of the position-lock cam.

As shown in the previous figure, the phase shift move affects neither the actual nor the command position of the master. Therefore, a phase shift move of one slave axis does not affect other axes slaved to the same master axis.

The figure below shows the motion profile of a position-lock cam and the resultant profile after execution of a phase shift Move Axis block and an incremental move on the slave axis (for comparison).



Note that if the master axis is not moving, you can achieve this same effect using another position-lock cam block with a new master reference position. The phase shift move, however, allows this adjustment to be made at a controlled rate while the master is moving and the position-lock cam is executing.

You can use phase shift moves to provide slip compensation for the master axis when you use a position-lock cam. In such a case, the amount of slip can be determined using the standard registration capability. A phase shift Move Axis block can smoothly adjust the position-lock cam profile by the slip distance.

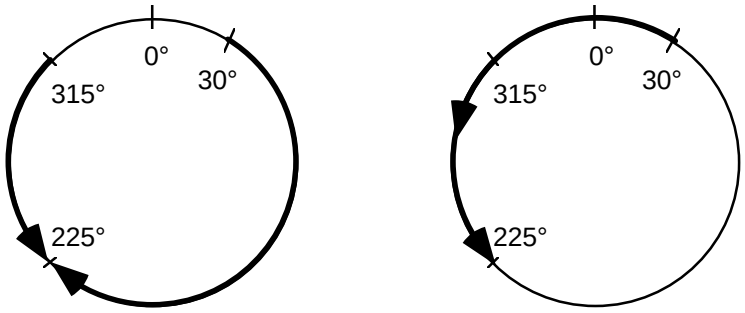
Rotary Shortest Path Moves

Rotary Shortest Path moves a rotary Axis to the desired absolute position via the shortest path. It is a special type of absolute move available only for rotary axes.

ATTENTION: Use Rotary Shortest Path moves only with rotary axes, and never with linear axes.

When you select Rotary Shortest Path, the axis moves to the specified position at the specified speed (using the specified acceleration and deceleration) in the direction that results in the shortest move, regardless of the current position of the axis. The position must be a positive value less than the unwind value. Therefore, you cannot perform moves of more than one revolution (that is, more than 360 degrees) with a single rotary shortest path Move Axis block.

For example, assume you plan to move a rotary axis, with position units of degrees, to a position of 225°. As shown in the figure below, using the standard absolute Move Axis block, the direction of travel depends on the current position of the axis, and is not necessarily the shortest path to the endpoint. Starting positions less than the endpoint cause motion in the positive direction, while starting positions greater than the endpoint cause motion in the negative direction.



Standard Absolute Move

Rotary Shortest Path Move

With the rotary shortest path move however, the axis moves to the specified endpoint—through 0° if necessary—in the direction that results in the shortest move regardless of the current axis position.

You can use the rotary shortest path move while the axis is moving or standing still.

Rotary Positive Moves

Use the Rotary Positive Move to move a rotary Axis to the desired absolute position in the positive direction. It is a special type of absolute move available only for rotary axes.

ATTENTION: Use Rotary Positive moves only on rotary axes, and never on linear axes.

When you select Rotary Positive, the axis moves to the specified position at the specified speed (using the specified acceleration and deceleration) in the positive direction, regardless of the current position of the axis. The position must be a positive value less than the unwind value. Therefore moves of more than one revolution (that is, more than 360 degrees) cannot be performed with a single rotary positive Move Axis block.

For a rotary positive move the axis moves to the specified endpoint position—through 0° if necessary—in the positive direction regardless of the current axis position. Use the rotary positive move only while the axis is not moving, to ensure motion in the proper direction.

Rotary Negative Moves

Select Rotary Negative to move a rotary axis to the desired absolute position in the negative direction. It is a special type of absolute move available only for rotary axes.

ATTENTION: Use Rotary Negative moves only on rotary axes, and never on linear axes.

When you select Rotary Negative, the axis moves to the specified position at the specified speed using the specified acceleration and deceleration in the negative direction, regardless of the current position of the axis. The position must be a positive value less than the unwind value. Therefore, moves of more than one revolution (that is, more than 360 degrees) cannot be performed with a single rotary negative Move Axis block.

With the rotary negative move the axis moves to the specified endpoint position—through 0° if necessary—in the negative direction regardless of the current axis position. Use the rotary negative move only while the axis is not moving to ensure motion in the proper direction.

Override Profile

You can also use the Move Axis block to override the power-up move profile selection, and move the axis using a different velocity profile. Select Override Profile and select both the desired axis and motion profile—Trapezoidal, S-Curve, or Parabolic. Enter values or expressions for the desired position or distance, speed, acceleration, and deceleration. Speed, acceleration, and deceleration values can be entered either as percentages of the current 100% values, or directly in the position units of the axis. See the *Motion Settings* section of the *Control Setting Blocks* chapter for information on the three types of move profiles.

Merged Moves

You can also use absolute or incremental moves to terminate the motion produced by a previous Jog Axis, Position Lock Cam, Time Lock Cam, or Gear Axes block. The axis stops either at a specified position, or after a specified distance. Select Merge from Jog, Cam, or Gear, and select the desired axis and type of move—absolute or incremental only—and the speed of merge (see the following paragraphs). Enter values or expressions for the desired position or distance, speed, acceleration, and deceleration. Select other options as needed.

Current Speed

Selecting At Current Speed from the Merge from Jog, Cam, or Gear menu, automatically sets the speed of the move to the current actual speed of the axis. Any speed value or expression is ignored. Under certain conditions, the speed of the merged move may not exactly match the current speed of the axis. See the *Merging Different Types of Motion* chapter in this manual for a discussion of this situation.

Programmed Speed

Selecting At Programmed Speed from the Merge from Jog, Cam, or Gear menu, sets the speed of the move to the speed value or expression entered in the *Speed* field of the Move Axis block. If this speed is different from the current speed of the axis, the axis accelerates or decelerates to the new speed.

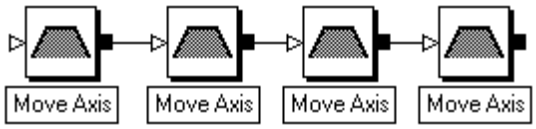
Wait for Completion

If you select Wait for completion, the motion controller halts the execution of other program blocks until the current move is done and there are no faults on the axis. When the move is done, the program continues with the next block. If other tasks are executing (multitasking), the motion controller halts the task that contains the current move block, while all other tasks continue to execute. In this way, a Wait for completion selection in one task does not halt execution of any other tasks, or hang the task dispatcher.

If you do not select Wait for completion, the motion controller begins the Move Axis block's programmed motion, and then proceeds to execute the next block (or blocks) in the program.

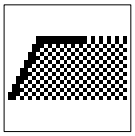
Synchronizing Moves

If you select Synchronize with next Move Axis, the motion of this Move Axis block occurs simultaneously with the following Move Axis block. This lets you initiate motion on multiple axes simultaneously, as shown below.



When you select Synchronize with next Move Axis, the block immediately following the Move Axis block must be another Move Axis block. Otherwise, GML Commander does not let you connect the two function blocks. In the example above, you must select Synchronize with next Move Axis in the first three Move Axis blocks—but not in the last block—to start all four axes simultaneously.

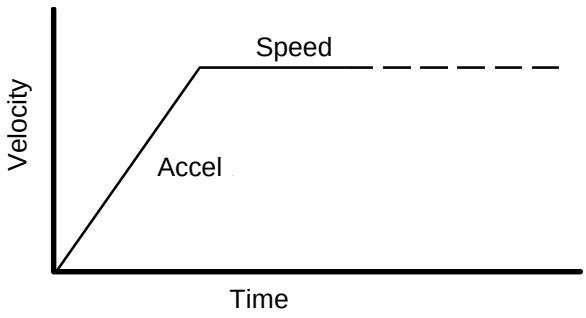
Jog Axis



The Jog Axis block jogs (continuously moves) a servo axis in a specified direction, at a specified speed, using specified acceleration and deceleration values.

The Jog Axis function block resides on the Main Palette.

The figure below shows the general form of a trapezoidal jog starting with the axis at standstill.



While a jog is in progress:

- Jog_status = 1 (true).

- Axis_status = 2 (Jogging).
- Lock_status = 0 (Unlocked).

When the jog is done:

- Jog_status = 0 (false).
- Axis_status = 1 (Unlocked).

For physical axes, when the jog is done and the position error of the axis is less than the position lock tolerance:

- Axis_status = 0 (Locked).
- Lock_status = 1 (Locked).

See the *Axis Locked and Axis Done Conditions* chapter of this manual, for a complete discussion of the difference between these conditions. See the *System Variables* chapter in this manual for more information on variables.

Override Profile

You can also use the Jog Axis block to override the power-up jog profile selection, and jog the axis using a different velocity profile. Select Override Profile and select both the desired axis and motion profile—trapezoidal, S-curve, or parabolic. Enter values or expressions for the desired direction, speed, acceleration and deceleration. Speed, acceleration, and deceleration values can be entered either as percentages of the current 100% values, or directly in the position units of the axis. See the *Motion Settings* section of the Control Setting Blocks chapter for information on the three types of move profiles.

Merged Jogs

You can also use jogs to terminate the motion produced by a previous Position Lock Cam, Time Lock Cam, or Gear Axes block, with the result that the axis continues moving at the specified speed. Select Merge from Cam, or Gear, and select the desired axis and type of merge—At Current Speed, or At Programmed Speed (see the following paragraphs). Enter values or expressions for the desired speed, acceleration, and deceleration.

Current Speed

Selecting At Current Speed from the Merge from, Cam, or Gear menu, automatically sets the speed of the jog to the current actual speed of the axis. Any speed value or expression is ignored. Under certain conditions, the speed of the merged move may not exactly match the current speed of the axis. See the *Merging Different Types of Motion* chapter in this manual for a discussion of this situation.

Programmed Speed

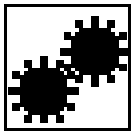
Selecting At Programmed Speed from the Merge from, Cam, or Gear menu, sets the speed of the move to the speed value or expression entered in the *Speed* field of the Jog Axis block. If this speed is different from the current speed of the axis, the axis accelerates or decelerates to the new speed.

Synchronize with next Jog Axis

Select Synchronize with next Jog Axis to have the motion of this Jog Axis block occur simultaneously with the following Jog Axis block. This lets you initiate motion on multiple axes simultaneously.

When you select Synchronize with next Jog Axis, the block immediately following the Jog Axis block must be another Jog Axis block. Otherwise, GML Commander does not let you connect the two function blocks.

Gear Axes



The Gear Axes block enables electronic gearing between two axes at a specified ratio. Electronic gearing allows any servo axis to be synchronized to the actual or command position of another axis at a precise ratio. It provides a direct edge-to-edge lock between the two axes—no maximum velocity, acceleration, or deceleration limits are used. The speed, acceleration, and deceleration of the slave axis are completely determined by the motion of the master axis and the specified gear ratio.

The Gear Axes block resides on the Main Palette.

When electronic gearing is enabled:

- Gearing_status = 1 on the slave axis.
- Axis_status = 5 if no faults are active on the slave axis.

Electronic gearing remains active through any subsequent Jog Axis, Move Axis, or Time Lock Cam blocks for the slave axis. This lets you superimpose electronic gearing motions on the jog, move, or time-lock cam profile motions to create complex motions and synchronization. See the *Axis Locked and Axis Done Conditions* chapter of this manual, for a complete discussion of the operation of these status conditions when a move, jog, or time-lock cam is superimposed on gearing motion.

Virtual Master Axes Gearing

Because AxisLink virtual axes are functionally equivalent to physical Master Only axes, you can use a virtual axis as the master axis for electronic gearing. However, only one of the two available virtual axes can be enabled at one time (enabling one virtual axis automatically disables the other virtual axis, if it had previously been enabled). So, if you want to use an AxisLink virtual axis as the master axis for electronic gearing, you first must enable the virtual axis with a Virtual Axis Control block. See *Virtual Axis Control Block* in the *AxisLink Blocks* chapter for information on enabling virtual axes.

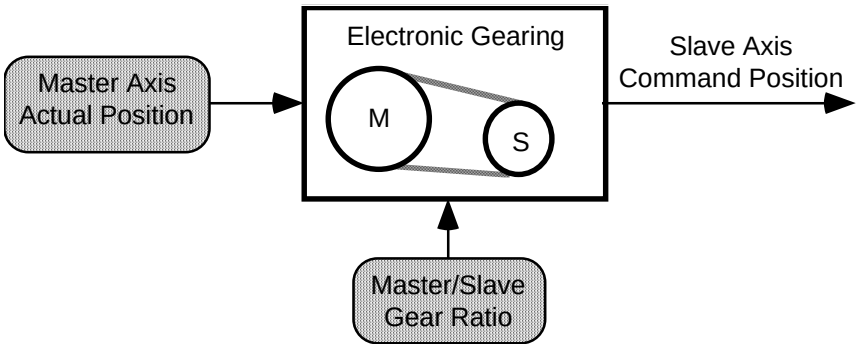
Because virtual axes are equivalent to physical master only axes, only Actual position is available on the Slave to menu.

Imaginary Axis Gearing

You also can use the imaginary axis for gearing, as either the master axis or the slave axis. Because the output of the imaginary axis is its command position (it has no actual position), only Command position is available on the Slave to menu.

Slaving to Actual Position

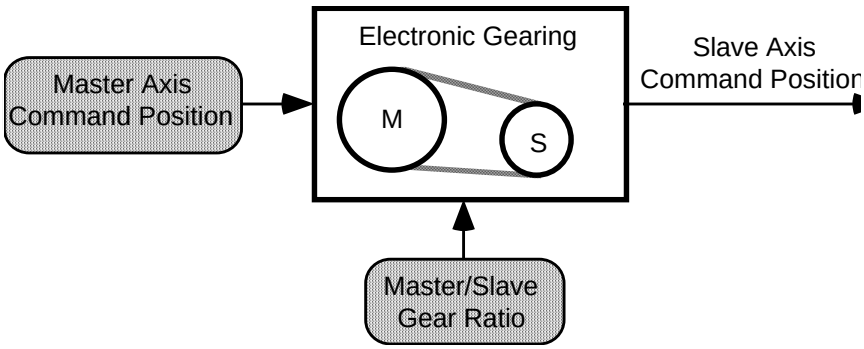
When you select Actual position from the Slave to menu, changes in the actual position of the master axis generates slave axis motion, as shown below.



Actual position is the current position of a physical or virtual master axis as measured by an encoder or other feedback device. This is the usual selection—and it is the only selection when the master axis is a master only (physical or virtual) axis—because it usually is the case that you must synchronize the actual positions of two axes.

Slaving to Command Position

When you select Command position from the Slave to menu, changes in the command position of the master axis generates slave axis motion, as shown below.



Command position (only available when the master axis is a servo axis) is the intended position for the master axis as commanded by any previous motion blocks. Because a virtual axis is equivalent to a physical master only axis, a virtual axis cannot be a slave axis.

Because the command position does not incorporate any associated following error or external position disturbances, it is a more accurate and stable reference for gearing. When gearing to the command position of the master, you must command the master axis to move in order to cause motion on the slave axis. See the *Installation and Setup* manual for your motion controller for more information on command position.

Same Direction Gearing

When you select Same from the Direction menu, the slave axis moves in its positive direction at the specified gear ratio when the master axis moves in its positive direction and vice-versa. If a new gear ratio is not specified (i.e., if you do not select Set Ratio), the last gear ratio specified for the slave axis is used. In most cases, this means that the current gear ratio is retained.

Opposite Direction Gearing

When you select Opposite from the Direction menu, the slave axis moves in its negative direction at the specified gear ratio when the master axis moves in its positive direction and vice-versa. If a new gear ratio is not specified (i.e., if you do not select Set Ratio), the last gear ratio specified for the slave axis is used. In most cases, this means that the current gear ratio is retained.

Changing the Gear Ratio (Unchanged Direction Gearing)

Selecting Unchanged from the Direction menu, lets you change the gear ratio while preserving (and not changing) the current gearing direction (same or opposite). This is useful when the current direction is not known or not important.

Reversing the Gearing Direction

When you select Reverse from the Direction menu, the current direction of the electronic gearing changes from same to opposite or from opposite to same. This is very useful for winding applications where the gear ratio must be reversed at each end of the wind. If you do not specify a new gear ratio (i.e., if you do not select Set Ratio), the motion controller uses the last gear ratio specified for the slave axis.

Set Ratio

Set Ratio is a required field for the first Gear Axes block in a program. It is an optional field for subsequent Gear Axes blocks, which can be used to change other Gear Axes function block parameters, while retaining the initial Slave: Master Ratio.

Real Number Gear Ratios

When you select Real in the *Set Ratio* menu, you define the gear ratio as a real number or expression. In the *Slave:Master Ratio* field, enter a positive number or an expression with a value between the following representing the desired ratio of slave axis position units to master axis position units.

0.00001 and 999,999.99999 (inclusive)

A gear ratio expressed this way is easy to interpret because it is defined in the axes' own position units.

If your gear ratio cannot be exactly expressed as a real number with a maximum of five digits to the right of the decimal point, select Fraction in the Set Ratio menu.

Fraction Gear Ratios

When you select Fraction from the Set Ratio menu, you define the gear ratio as a pair of integer numbers or expressions representing the ratio between the following:

- the number of slave axis feedback counts
- the number of master axis feedback counts

You can use up to five digits (1 to 99,999) for the slave counts, and up to nine digits (1 to 999,999,999) for the master counts.

Specifying the gear ratio as a fraction allows the use of irrational numbers (such as 1/3) as gear ratios, with no accumulated positioning errors or round off. Because the master and slave count values do not use the axis conversion constants, and because they are integers, the actual gear ratio relationship between the slave and master axes exactly matches the specified ratio.

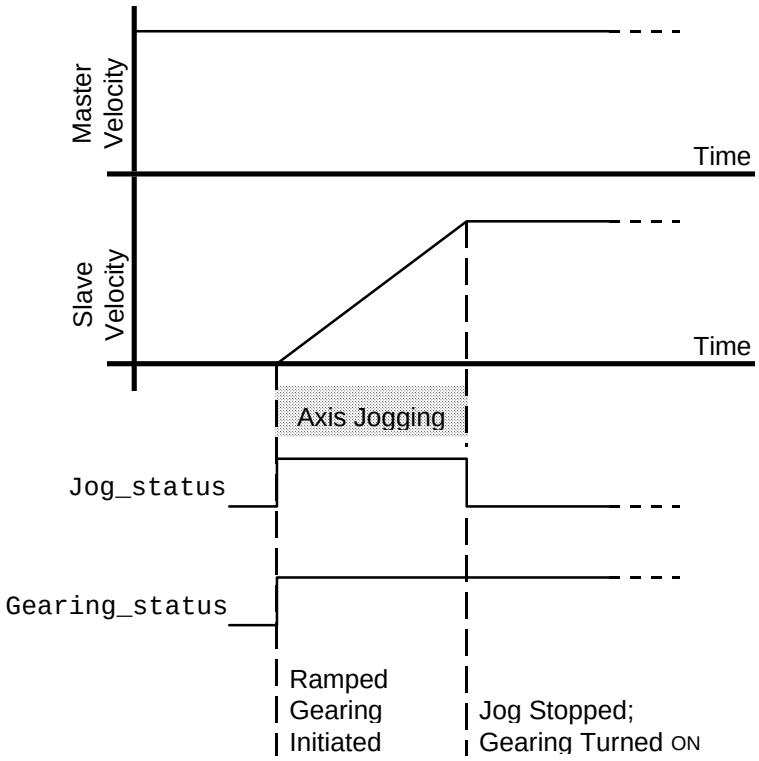
For example, the irrational gear ratio 1/3 can be specified as 1 slave count to 3 master counts, 10 slave counts to 30 master counts, 3 slave counts to 9 master counts, etc.

Ramping to Master Speed

If you select Ramp to Master Speed, the slave axis accelerates or decelerates to gearing speed (i.e., to the speed the slave axis would be moving if it were geared to the selected master axis, given the specified gearing ratio) using a trapezoidal profile. This causes linear acceleration or deceleration. Once the slave axis has reached gearing speed, electronic gearing is automatically activated according to the other selections.

Enter a value or expression for the acceleration rate, either as a percentage of the maximum acceleration rate, or in the slave axis position units per second^2.

Ramping provides an electronic clutch that smoothly engages the slave axis to the master axis, as shown below.

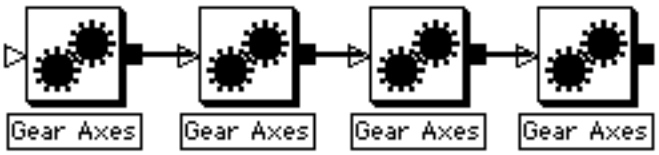


The command position, of both the slave and master axes, is stored as the Strobed_Position when you initiate ramped gearing.

Ramped gearing avoids the uncontrolled acceleration or deceleration caused by enabling electronic gearing while the master axis is moving. You can also use ramping to merge gear ratio changes on-the-fly—even changes in direction. The motion controller automatically ramps the slave axis to the gearing speed commanded by gearing it to the master axis at the new ratio and/or direction.

Synchronizing Gearing on Multiple Axes

When you select Synchronize with next Gear Axes, the current Gear Axes block executes simultaneously with the following Gear Axes block. This lets you initiate electronic gearing on multiple axes simultaneously.



When you select Synchronize with next Gear Axes, the block immediately following the current Gear Axes block must be another Gear Axes block, otherwise GML Commander does not let you connect the two blocks. In the example above, Synchronize with next Gear Axes should be selected in the first three Gear Axes blocks—but not in the last block—to start all four axes simultaneously.

Using the Gear Axes Block

Changing Master Axes

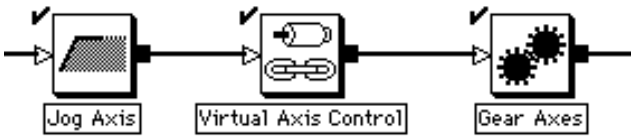
You can change the master axis for electronic gearing at any time, even while gearing is currently enabled. But be careful, because electronic gearing can be enabled on more than one axis at a time. When a servo master axis and slave axis are reversed, the axes become cross-coupled and unexpected motion may result.

For example, if you are gearing Axis 0 to Axis 1 (defined as a Servo axis) and then decide to change to gearing Axis 1 to Axis 0, you must first disable gearing on Axis_0. This is because specifying Axis 1 as the slave axis with Axis 0 as the master axis does not automatically disable Axis 0 from being a slave axis with Axis 1 as the master axis. See the *Disable Gearing* section in this chapter.

ATTENTION: Changing virtual master axes on the fly can result in a temporary stoppage of motion on the Master Axis.

Changing virtual master axes on the fly is similar to changing physical Master Only master axes, but with one exception. When the new virtual axis is enabled, the other virtual axis is automatically disabled. Because enabling a virtual axis can take up to 10 milliseconds, there can be a short period of time when no master is active. If the slave axis is moving, this produces a discontinuity in its motion.

To eliminate this problem, merge the existing gearing motion to a jog, enable the new virtual axis, and then merge the jog to gearing with the new master axis, as shown below.



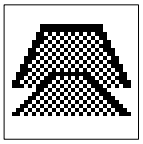
In the Jog Axis block, select Merge from Cam or Gear and select At Current Speed from the menu. This causes the slave axis to continue moving at its current speed and turns gearing off. Enable the desired new virtual master axis in the Virtual Axis Control block and select Wait for Linked. In the Gear Axis block, select the newly enabled virtual axis as the master axis, and select Ramp to Master Speed. This causes the slave axis to ramp smoothly to its new gearing speed. See the *Virtual Axis Control Block* in the *AxisLink Block* chapter for more information.

Moving While Gearing

You can use an incremental Move Axis block for the slave axis (or master axis if configured for servo operation) while the electronic gearing is enabled. This is particularly useful to accomplish phase advance/retard control. You can use the incremental move distance to eliminate any phase error between the master and the slave, or to create an exact phase relationship.

Normally a gear ratio of 1 is used with phase adjustment. A 1:1 ratio ensures that the computed phase error does not change before performing the move to correct it. Electronic gearing is not normally used with absolute moves, because the ultimate endpoint is not predictable.

Interpolate Axes



Use the Interpolate Axes block (for motion controllers with iCODE version 3.0 or later) to move a group of two or more servo axes:

- to a specified absolute position or by a specified incremental distance
- along a specified linear, circular, or helical path
- at a specified speed
- at a specified rate of acceleration and deceleration
- using a selected motion profile

You can move up to four axes simultaneously using linear interpolation. You can move two axes with circular interpolation, and up to three axes with helical interpolation.

The Interpolate Axes block resides on the Advance Motion Palette.

The Interpolate Axes function block parameters vary, depending upon which type of interpolated motion you select. There are four interpolated motion types:

- Linear
- Radius Arc
- Intermediate Arc
- Helical

The parameters for each type of motion are set forth and discussed later in this section.

While an interpolated move is in progress:

- Interp0_status or Interp1_status = 1 (true) depending on the selected interpolator
- Axis_status ≤ 5 for each selected axis if no faults are active on the axis.

When the interpolated move is done:

- Interp0_status or Interp1_status = 0 (false).

See the *Axis Locked and Axis Done Conditions* chapter in this manual for more information on axis status during interpolation. See the *System Variables* chapter in this manual for more information on variables.

Profile

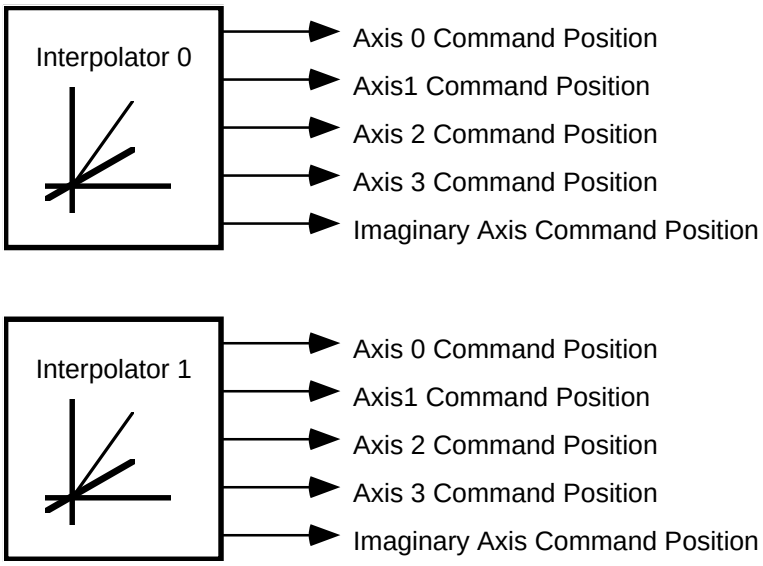
The vector acceleration and deceleration of interpolated moves can be either a trapezoidal or an S-curve profile. The S-curve profile selection is available only on controllers using iCODE versions 3.6 or higher.

The deceleration rate is always the same as the acceleration rate and cannot be changed while the interpolated move is in progress. Interpolated moves work best when all axes use the same position units.

Important: Before executing an interpolated move, be sure to set all axes to the same Transducer Resolution Conversion Constant (expressed in counts per axis units), in the Feedback page of the Configure Axis Use dialog box.

Interpolators

The motion controller provides two separate interpolators (Interpolator 0 and Interpolator 1) which are used by the Interpolate Axes block for moving groups of axes together. Each interpolator has five outputs—one for each of the four possible physical axes and one for the imaginary axis on the motion controller, as shown below.

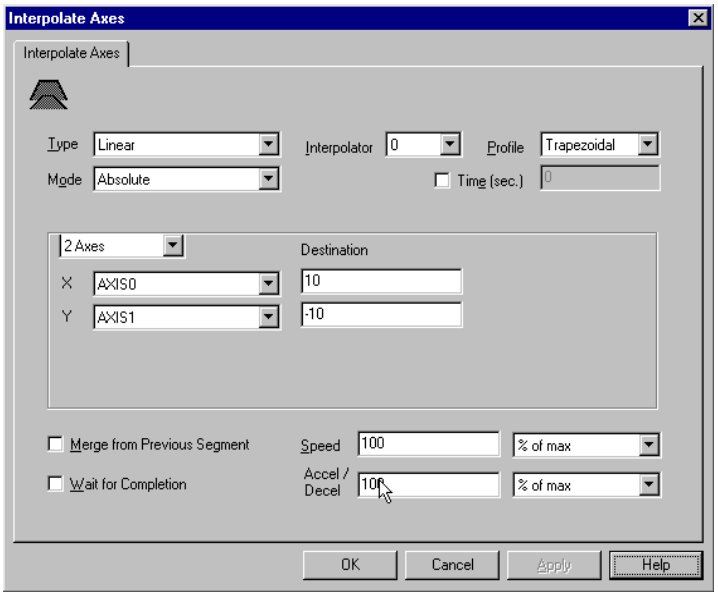


Like all other high-level motion functions in the motion controller, the interpolator outputs only contribute to the command position of each axis. Thus, other types of motion (gearing, cams, etc.) can be added to the interpolated motion for special applications. In addition, the two interpolators may be used concurrently with the same or different axes to combine two interpolated motions on a set of axes.

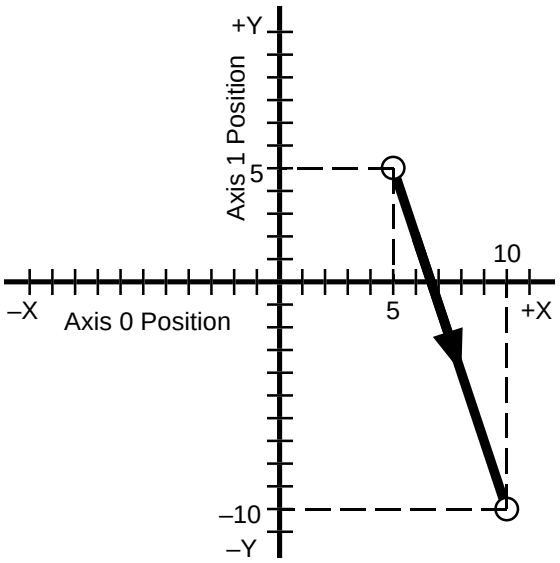
Linear Interpolation

When you select Linear from the Type menu, the selected axes move the specified distances (in Incremental mode) or move to the specified destination positions (in Absolute mode).

For example, the Interpolate Axes parameters shown below, use Interpolator 0 to move Axis 0 to absolute position 10 (position units) and Axis 1 to absolute position –10 (position units) at the current 100% velocity using the current 100% acceleration and deceleration values.



If the axes are orthogonally disposed to each other, and are both currently at absolute position 5, the motion caused by this block is as shown below.



The vector speed of the selected axes (assuming an orthogonal Cartesian disposition of axes) is equal to the specified speed in the position units of the X axis per second or percent of the maximum speed of the X axis. The speed of each axis is proportional to the distance traveled by the axis divided by the square root of the sum of the squares of the distances moved by all axes. Continuing our example above, the actual speed of Axis 0 is the following percent of the vector speed of the move (the current 100% speed of Axis 0).

$$\frac{(10-5)}{\sqrt{(10-5)^2+(-10-5)^2}} = \frac{5}{\sqrt{5^2+(-15)^2}} = 0.3162 = 31.62\%$$

Likewise, the vector acceleration and deceleration (linear) is equal to the specified acceleration/deceleration in the position units of the X axis per second^2 or percent of the maximum acceleration of the X axis. The acceleration and deceleration of each axis is proportional to the distance traveled by the axis divided by the square root of the sum of the squares of the distances moved by all axes. The deceleration rate is always the same as the acceleration rate and cannot be changed while the move is in progress.

Timed Linear Interpolation

When you select both Linear (from the Type menu) and Time, the selected axes move the specified distances (in Incremental mode) or to the specified destination positions (in Absolute mode) in the specified time (in seconds).

In timed linear interpolation moves, the time and distance values you input are used to calculate the interpolated move's speed, acceleration, and deceleration values. The speed and acceleration/deceleration parameter values that you input (in the *Speed* and *Acceleration/ Deceleration* fields) are used as the maximum values for the interpolated move. Provided that the calculated acceleration/deceleration value does not exceed the input acceleration/deceleration maximum value, and the calculated velocity does not exceed the input speed value, the axes will move to the commanded position.

If the calculated acceleration/deceleration value exceeds the input acceleration/deceleration value, or the calculated velocity exceeds the input speed, an Insufficient Time runtime fault (Runtime_fault = 28) is generated when you execute the block. You can correct this problem if you increase the move's:

- time setting
- maximum speed setting
- maximum acceleration/deceleration setting.

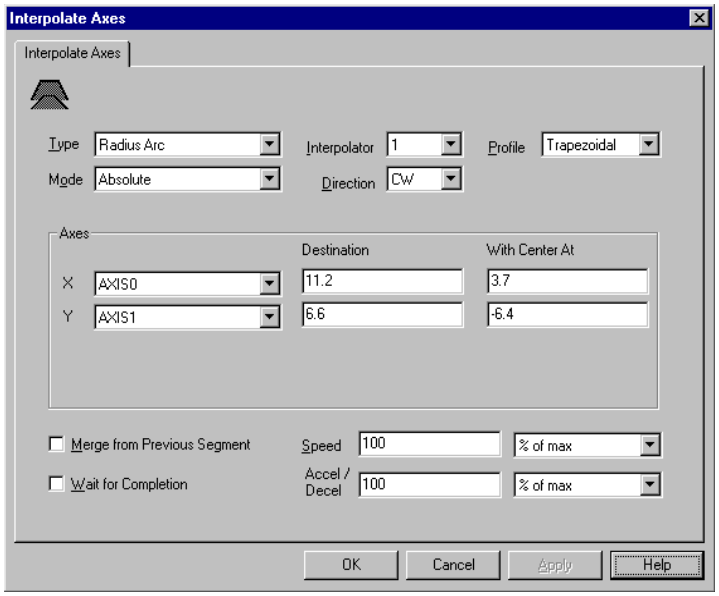
See *Runtime Fault* under the *System Variables* chapter in this manual for more information on runtime faults.

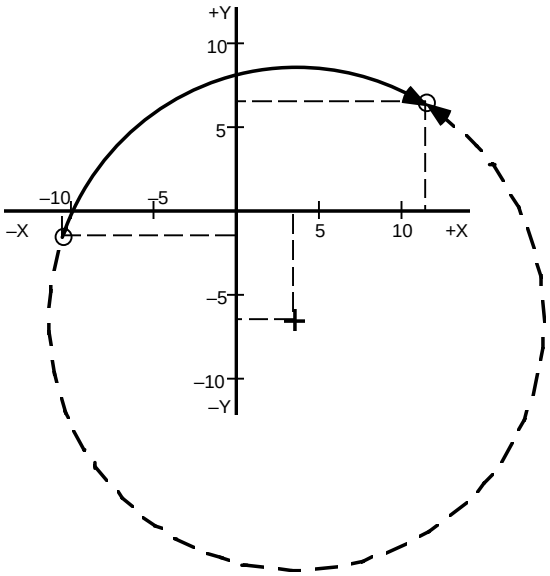
Radius Arc Circular Interpolation

When you select Radius Arc from the Type menu, the selected axes move the specified distances (in Incremental mode), or to the specified destination positions (in Absolute mode), in the commanded direction, along the perimeter of a circle with its center at a specified point.

Example 1: Radius Arc Absolute Mode

For example, if Axis 0 is the X axis and is currently located at absolute position –10.4 position units, and Axis 1 is the Y axis and is currently located at position –1.3 position units, the Interpolate Axes parameters shown below use Interpolator 1 to move Axis 0 to an endpoint of 11.2 and Axis 1 to an endpoint of 6.6 along a circular arc whose center is at X = 3.7, Y = –6.4 in a clockwise direction, as shown by the solid line below.





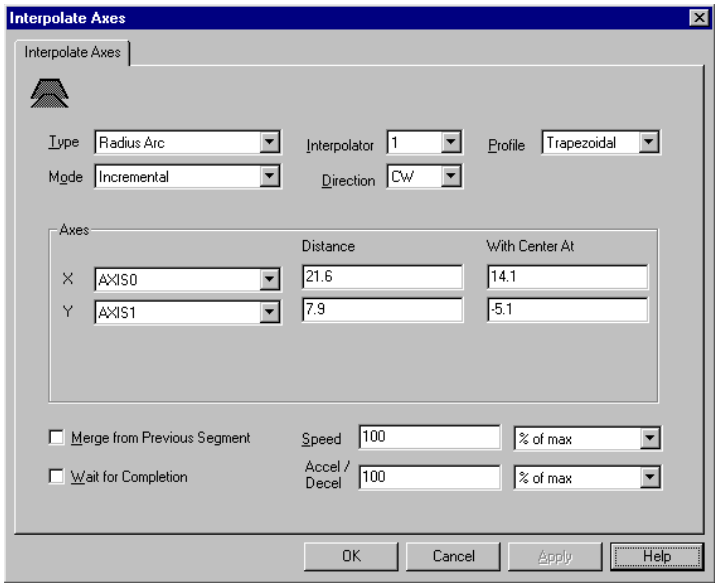
If you had, instead, selected CCW from the Direction menu, the axes would have moved along the arc shown by the dashed line above.

Example 2: Radius Arc Incremental Mode

If you are operating in incremental mode, both the end point and center point of the arc are defined in terms of their distance from the current position.

If the following bullets are true, and the Interpolate Axes parameters are as shown in the dialog box below, it produces the same motion as shown previously for absolute mode.

- Axis 0 is the X axis and is currently located at absolute position –10.4 position units
- Axis 1 is the Y axis and is currently located at position –1.3 position units

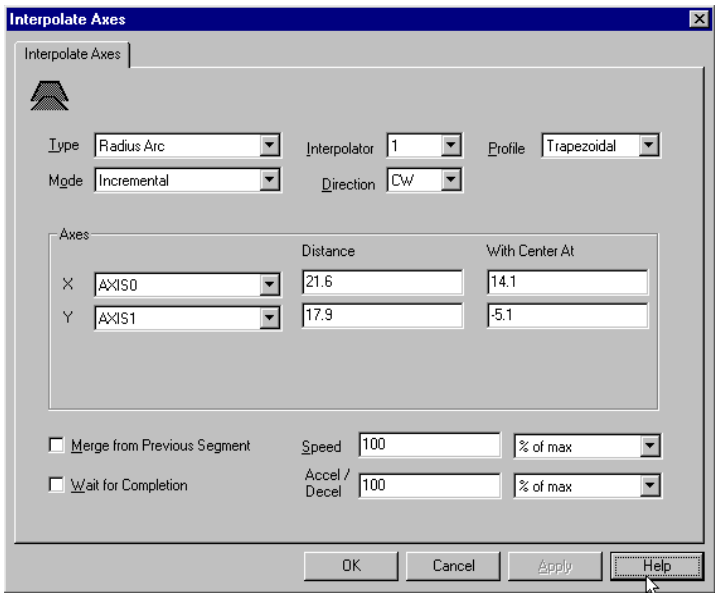


The vector speed of the selected axes is equal to the specified speed in the position units of the X axis per second or percent of the maximum speed of the X axis. Likewise the vector acceleration and deceleration (trapezoidal profile) is equal to the specified acceleration/deceleration in the position units of the X axis per second^2 or percent of the maximum acceleration of the X axis. The deceleration rate is always the same as the acceleration rate and cannot be changed while the move is in progress.

Bad Arcs

Unfortunately, radius arc circular interpolation allows you to enter dimensions that are non-commensurate—values that do not describe a real circle or arc. Interpolate Axes blocks of this nature cause a Bad Arc runtime fault (Runtime_fault = 27) when you execute the program. See *Runtime Fault* under the *System Variables* chapter in this manual for more information on runtime faults.

For example, the Interpolate Axes block parameters, shown below, cause a Bad Arc runtime fault when you execute the block, because the arc endpoint ($\Delta X = 21.6$, $\Delta Y = 17.9$) does not lie on the circle whose center is ($\Delta X = 14.1$, $\Delta Y = -5.1$).



The most frequent cause of bad arcs, other than typographical errors, is incorrect trigonometric calculations, or lack of sufficient precision in the calculations.

In the motion controller, the bad arc check is implemented by calculating the radius of the arc at both the start and end points. If the radius of the arc at the end point differs from the radius of the arc at the start point by more than ±0.5%, the arc is deemed a bad arc.

In Absolute mode, the radius of the arc at the start point is as shown in the first equation below and the radius of the arc at the end point is as shown as in the second equation below.

$$\sqrt{(\text{Command Pos. X} - \text{Center X})^2 + (\text{Command Pos. Y} - \text{Center Y})^2}$$

$$\sqrt{(\text{Destination X} - \text{Center X})^2 + (\text{Destination Y} - \text{Center Y})^2}$$

In Incremental mode, the radius of the arc at the start point is as shown in the first equation below, and the radius of the arc at the end point is as shown in the second equation below.

$$\sqrt{(\text{Center X})^2 + (\text{Center Y})^2}$$

$$\sqrt{(\text{Destination X} - \text{Center X})^2 + (\text{Distance Y} - \text{Center Y})^2}$$

If you get a Bad Arc runtime fault (Runtime_fault = 27), use the formulas above to calculate the start and end radius of the arc. Then change the destination or distance values in the Interpolate Axes block as required to ensure that these two radii are within ±0.5% of each other. Note that ±0.5% is equivalent to three significant digits in the arc parameters.

Intermediate Arc Circular Interpolation

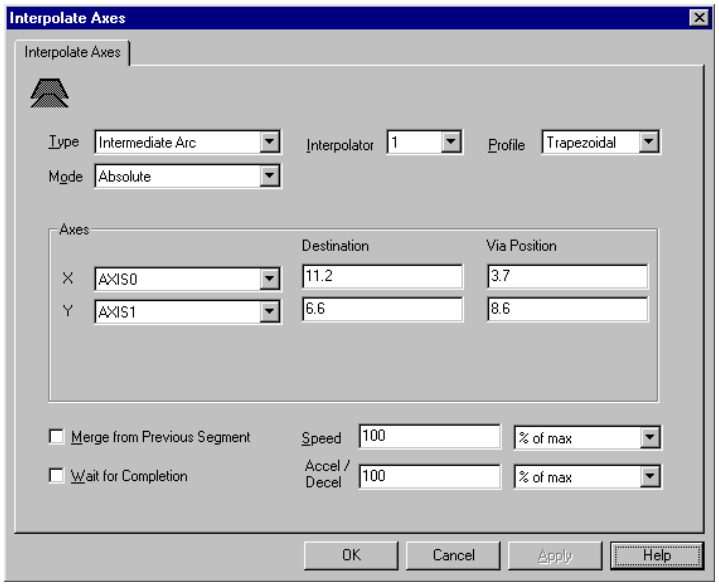
When you select Intermediate Arc from the Type menu, the selected axes move the specified distances (in Incremental mode), or to the specified destination positions (in Absolute mode), along a circular path which passes through the specified point.

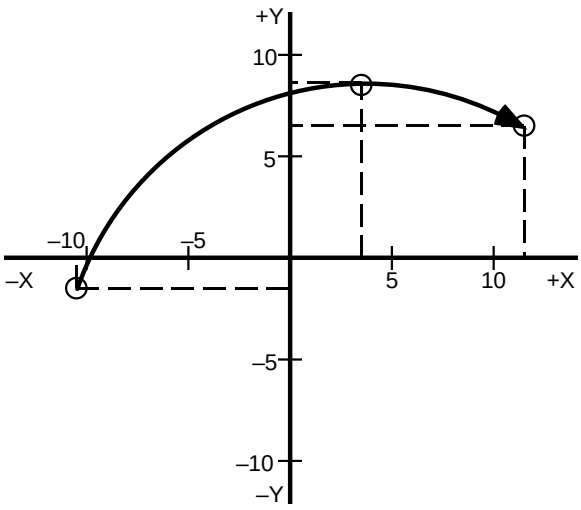
The vector speed of the selected axes is equal to the specified speed in the position units of the X axis per second or percent of the maximum speed of the X axis. Likewise the vector acceleration and deceleration (trapezoidal profile) is equal to the specified acceleration/deceleration in the position units of the X axis per second^2 or percent of the maximum acceleration of the X axis. The deceleration rate is always the same as the acceleration rate and cannot be changed while the move is in progress.

Example 3: Intermediate Arc Absolute Mode

For example, if the following bullets are true, and the Interpolate Axes parameters are as shown in the dialog box below, it uses Interpolator 1 to move Axis 0 to an endpoint of 11.2 and Axis 1 to an endpoint of 6.6 along a circular arc which passes through the point X = 3.7, Y = 8.6, as shown in the diagram below.

- Axis 0 is the X axis and is currently located at absolute position –10.4 position units
- Axis 1 is the Y axis and is currently at position –1.3 position units



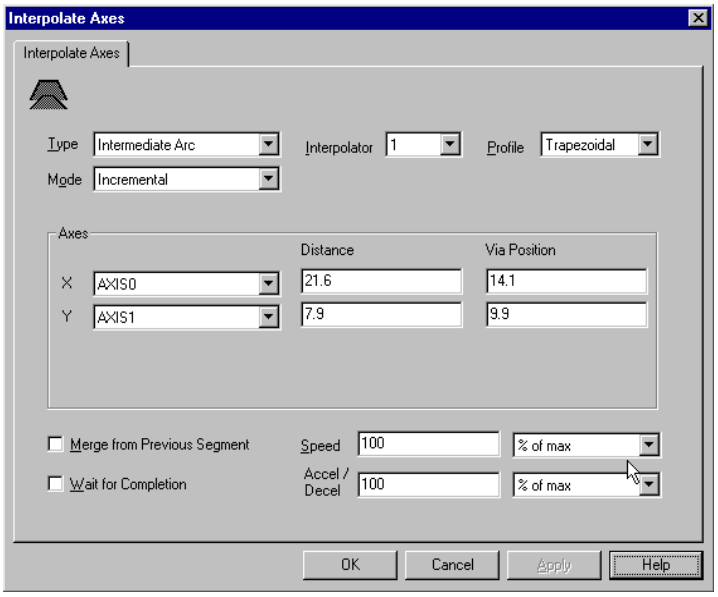


Example 4: Intermediate Arc Incremental Mode

If you are operating in incremental mode, both the end point of the arc, and the intermediate point through which the arc passes are defined in terms of their distance from the current position.

For example, if the following bullets are true, and the Interpolate Axes parameters are as shown in the dialog box below, it produces the same motion as shown previously for absolute mode.

- Axis 0 is used as the X axis and is currently at absolute position –10.4 position units
- Axis 1 is used as the Y axis and is currently at position –1.3 position units



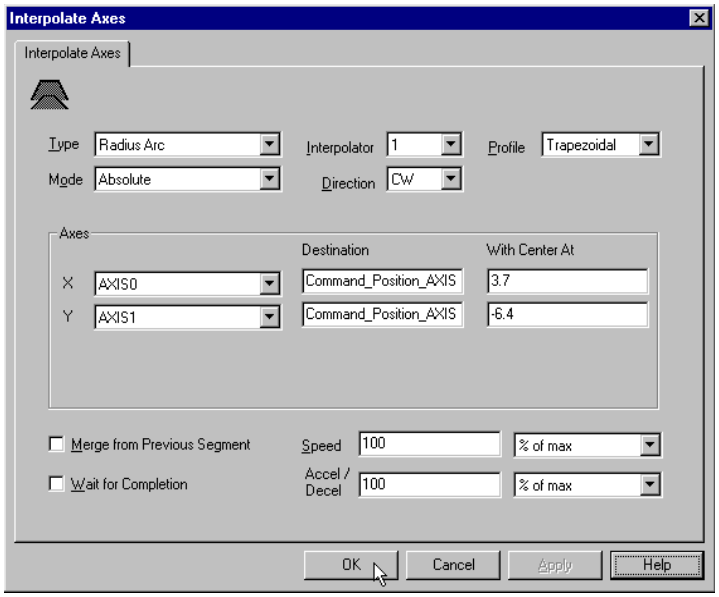
Since three points (the current position of the axes, the specified end point, and the specified intermediate point) define one and only one circle, it is difficult to program a bad intermediate arc. While it is still certainly possible to program an arc which is not the intended one, a Bad Arc runtime fault only occurs with intermediate arc circular interpolation if the three points are co-linear (all three are on the same line) or not unique (two or more are the same). In addition, the intermediate point implies the direction of the arc and thus it is not necessary to specify the direction.

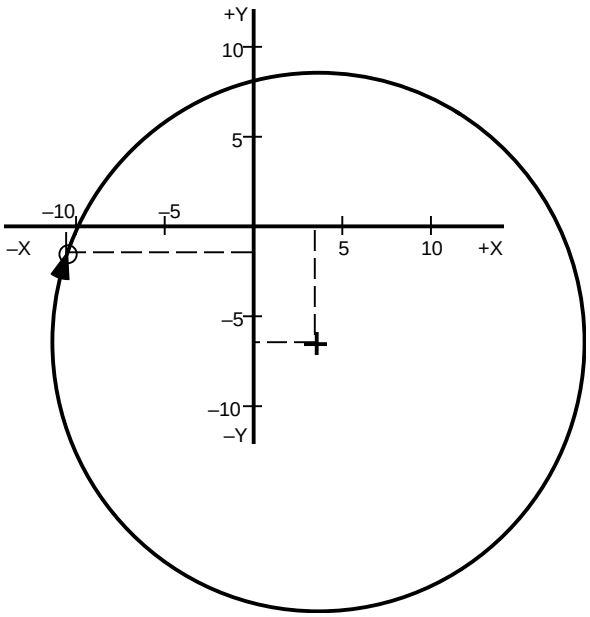
Full Circles

A complete circle is a special case of a circular arc in which the start and end point of the arc are the same.

Example 5: Full Circle in Absolute Mode

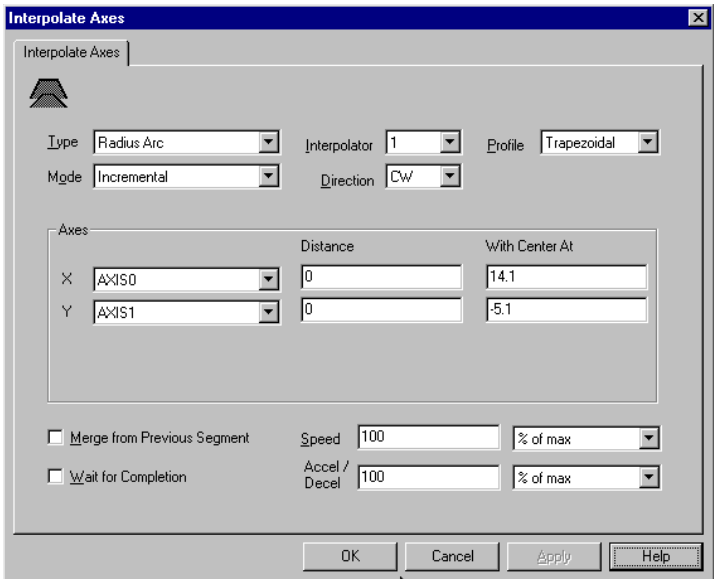
In Absolute Mode (using either Radius Arc or Intermediate Arc interpolation), the destination for each axis must be exactly equal to the command position of the axis at the beginning of the circle. Using the center point parameters of the previous example (Example 4*:* Intermediate Arc Incremental Mode), the Interpolate Axes parameters shown below use Interpolator 1 to move the axes in the following full circle as shown in the diagram below.





Example 6: Full Circle in Incremental Mode

In incremental mode, the distance for each axis must be exactly zero, as shown below:



Helical Interpolation

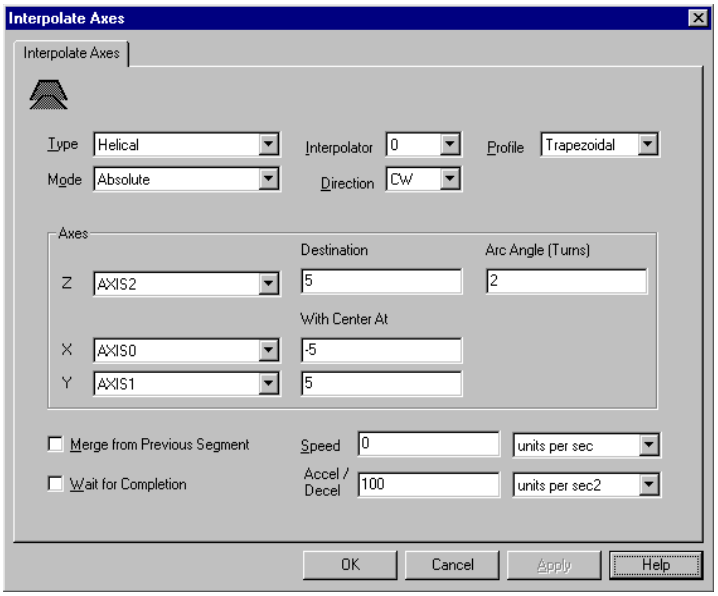
To enable helical interpolation, you must first enable the third and fourth physical axes (Axis2 and Axis3) in the General page of the Configure Control Options General dialog box.

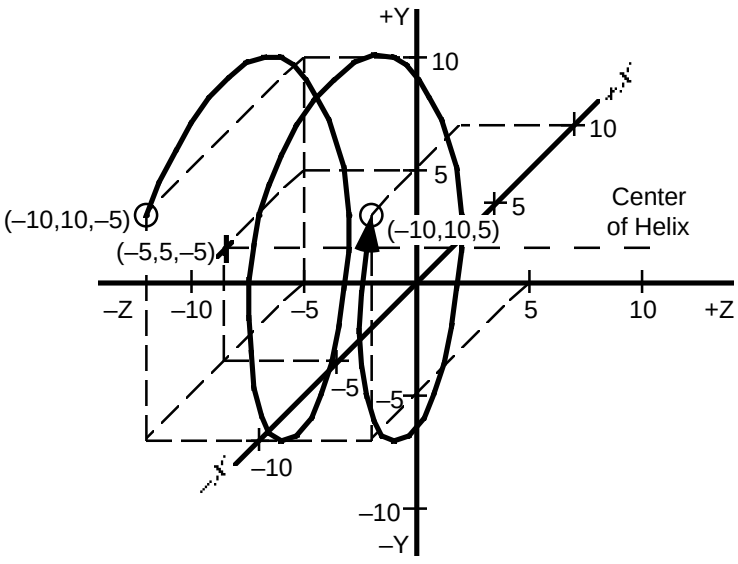
When you select Helical from the Type menu, the selected Z axis moves either the set distance (in Incremental mode), or to the specified destination position (in Absolute mode) while the selected X axis and Y axis move through the specified arc angle (in turns), in the selected direction, along a circular path whose center is at the set point, and produces the same motion as shown previously for absolute mode. This results in helical motion, a threading or spiraling of the three axes.

Example 7: Helical Interpolation in Absolute Mode

For example, if the following bullets are true, and the Interpolate Axes parameters are as shown below, it uses Interpolator 0 to move Axis 2 to an endpoint of 5 position units, while Axis 0 and Axis 1 make two complete revolutions around a circle centered at X = –5, Y = 5 in a clockwise direction, as shown in the diagram below.

- Axis 0 is used as the X axis and is currently at absolute position –10 position units
- Axis 1 is used as the Y axis and is currently at position +10 position units
- Axis 2 is used as the Z axis and is currently at position –5 position units


Interpolate Axes
Interpolate Axes
Type Helical
Interpolator 0
Profile Trapezoidal
Mode Absolute
Direction CW
Axes
Destination
Arc Angle (Turns)
Z AXIS2
5
2
With Center At
X AXIS0
-5
Y AXIS1
5
Merge from Previous Segment
Speed 0
units per sec
Wait for Completion
Accel / Decel 100
units per sec2
OK
Cancel
Apply
Help


+Y
10
5
–5
–10
–10
–Y
–Z
–10
–5
5
10
+Z
(–10,10,–5)
(–5,5,–5)
(–10,10,5)
Center
of Helix

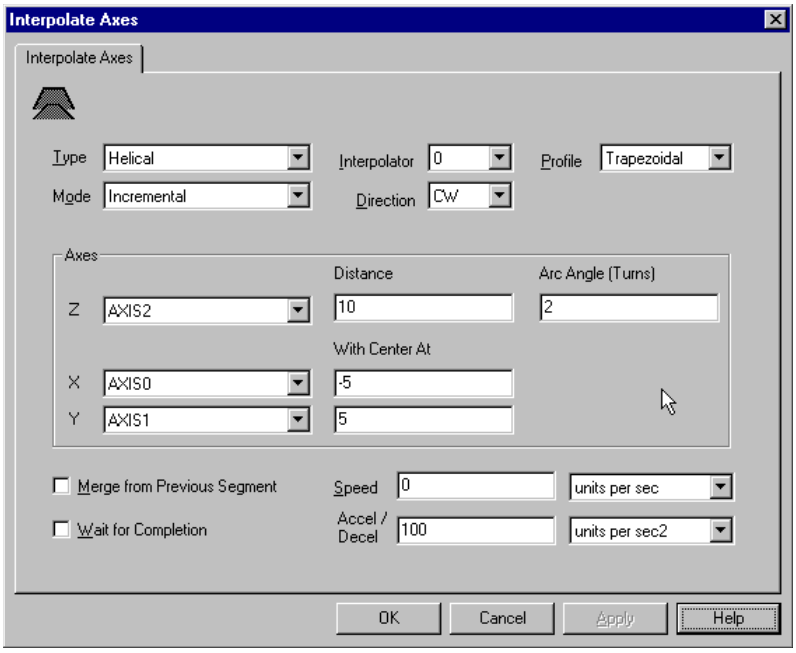
In this example, the absolute endpoints of the X and Y axes are the same as their start points, because an integral number of complete revolutions were made. If the specified arc angle is not an integer, the endpoints of the X and Y axes are not be the same as their start points.

Example 8: Helical Interpolation in Incremental Mode

In incremental mode, enter the distance from the current position of both the X and Y axes to the center of the circle in the *With Center At* fields. Then specify the helical move's distance along the Z axis, and the number of turns the X-Y plane will make around the Z axis.

For example, if the following bullets are true, and the Interpolate Axes parameters area as shown below, it produces the same motion as shown previously for absolute mode.

- Axis 0 is the X axis and is currently located at absolute position –10 position units
- Axis 1 is the Y axis and is currently located at position +10 position units
- Axis 2 is the Z axis and is currently located at position –5 position units



The radius of the helix is the vector distance from the current position of the X and Y axes to the specified center point. In Absolute mode, the radius of the helix is:

$$\sqrt{(\text{Command Pos. X} - \text{Center X})^2 + (\text{Command Pos. Y} - \text{Center Y})^2}$$

In Incremental mode, the radius of the helix is:

$$\sqrt{(\text{Center X})^2 + (\text{Center Y})^2}$$

The diameter of the helix is twice its radius.

The vector speed of the selected X and Y axes is equal to the specified speed in the position units of the X axis per second or percent of the maximum speed of the X axis. The speed of the selected Z axis is equal to the distance traveled by the Z axis divided by the total arc length of the helix times the specified speed. In absolute mode, the speed of the Z axis is the result of the equation below times the specified speed in the position units of the X axis per second or percent of the maximum speed of the X axis.

$$\frac{\text{Destination Z} - \text{Command Position Z}}{2\pi \times (\text{Helix Radius}) \times (\text{Arc Angle})}$$

In incremental mode, the speed of the Z axis is the result of the equation below times the specified speed in the position units of the X axis per second or percent of the maximum speed of the X axis.

$$\frac{\text{Distance Z}}{2\pi \times (\text{Helix Radius}) \times (\text{Arc Angle})}$$

Likewise the vector acceleration and deceleration (linear) of the axes is equal to the specified acceleration/deceleration in the position units of the X axis per second^2 or percent of the maximum acceleration of the X axis. The deceleration rate is always the same as the acceleration rate and cannot be changed while the move is in progress.

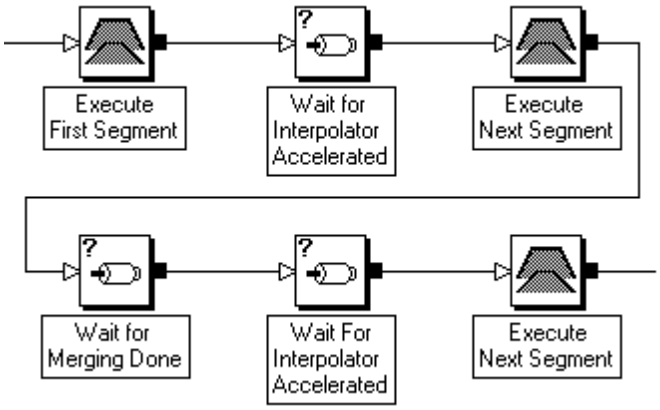
Merging Interpolated Motion

You can merge interpolated moves with previous motion from the same interpolator (using the same axes) as long as the segments are tangential at their intersection. To smoothly merge the motion caused by the current Interpolate Axes block with motion caused by a previous Interpolate Axes block, using the same interpolator and the same axes, select Merge from Previous Segment.

If the move segments are not tangential at their intersection, the axes come to a stop between the segments rather than merging smoothly. When merging linearly interpolated motions, only co-linear line segments (those that lie on the same line) can be smoothly merged with each other—any two line segments which are not co-linear are also not tangential.

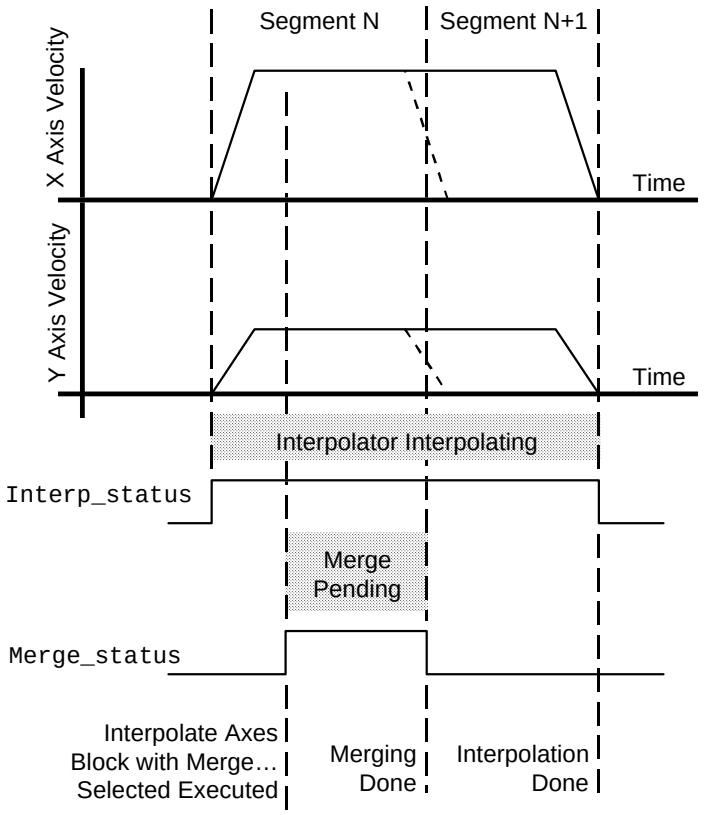
Also, do not execute an Interpolate Axes block with Merge From Previous Segment selected if the selected interpolator is accelerating, decelerating, or already has a merge pending, otherwise an Illegal Merge Attempt runtime fault (Runtime_fault = 31) occurs. See *Runtime Fault* in the *System Variables* chapter of this manual for more information on runtime faults.

The recommended GML Commander sequence for merging interpolated motion is shown below. Select Merge from Previous Segment in all Interpolate Axes blocks except the first.



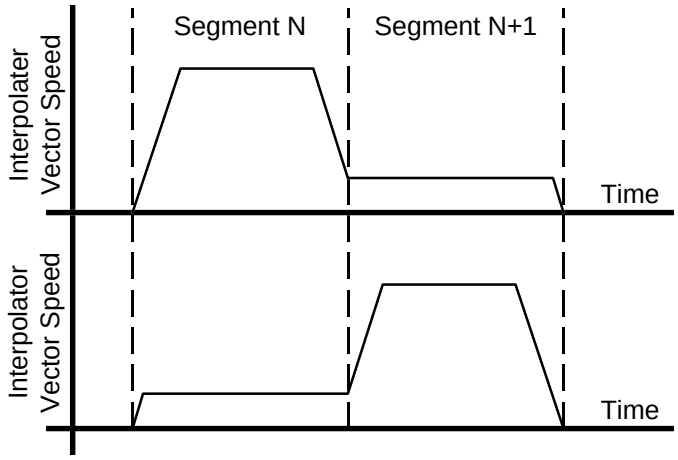
As shown above, the program must wait for the acceleration of the current segment to finish before executing the next Interpolate Axes block. For all merged segments except the first, the program waits for the current merge to finish before executing the next Interpolate Axes block.

After successfully executing an Interpolate Axes block with Merge from Previous Segment selected, Merge_status_Interp0 or Merge_status_Interp1 = 1 (depending on the selected interpolator) and the merged motion is pending. When the currently executing interpolated motion is done, Merge_status_Interp0 or Merge_status_Interp1 = 0. For example, the figure below shows the operation of the interpolator status and merge status variables when a two-axis linear move is merged into another two-axis linear move at the same speed.



The dotted line in the motion profiles above shows where Segment N would have ended if it had not been merged into Segment N+1.

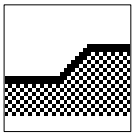
You can change the vector speed of a merged segment by specifying the desired new speed in an Interpolate Axes block with Merge from Previous Segment selected. If the specified vector speed of the new segment is less than the vector speed of the current segment, the axes decelerate to the new speed at the end of the current segment. Conversely, if the specified vector speed of the new segment is greater than the vector speed of the current segment, the axes accelerate to the new speed at the beginning of the next segment, as shown below.



Wait for Completion

When you select Wait for Completion, the program pauses until the selected motion finishes and there are no axes faults. When the motion has finished, the program continues with the next block. If other tasks are executing (multitasking), the task which contains this block pauses, but the other tasks continue to execute. In this way, Wait for Completion in one task does not halt execution of any other tasks or hang the task dispatcher. When you do not select Wait for Completion, the Interpolate Axis block merely initiates the selected motion, and the program continues by executing the subsequent blocks.

Change Dynamics



The Change Dynamics block changes the following for a servo axis, on-the-fly:

- speed of trapezoidal profile moves
- speed, acceleration, and deceleration of trapezoidal profile jogs

The Change Dynamics function block resides on the Main Palette.

Changing Move Dynamics

When you select Move in the *For current* field, you can change the speed of a move in progress by entering a new speed value. You cannot change the acceleration or deceleration values for a move in progress—only its speed. The axis ramps up to the new speed at the current acceleration rate (if the new speed is higher than the current speed), or ramps down to the new speed at the current deceleration rate (if the new speed is lower than the current speed).

ATTENTION: The Change Dynamics block affects only trapezoidal profile moves. Other velocity profiles cannot be changed on the fly, except to zero speed.

You can change trapezoidal profile moves to any speed at any time. However, you can change parabolic and S-curve profile moves only to zero speed, and then only if the axis is not decelerating (Decel_status = 0).

Pausing Moves

You can use the Change Dynamics block to temporarily pause a move in progress, by changing its speed to zero. Use another Change Dynamics block with a non-zero speed value to complete the move as originally specified.

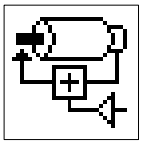
Changing Jog Dynamics

Selecting Jog in the *For current* field, lets you change the speed, acceleration, and/or deceleration of a jog in progress to the specified value. The speed change occurs at the specified acceleration rate (if the new speed is higher than the current speed), or at the specified deceleration rate (if the new speed is lower than the current speed).

Important: The Change Dynamics block affects only trapezoidal profile jogs. Other velocity profiles cannot be changed on the fly, except to zero speed.

.You can change trapezoidal profile jogs to any speed at any time. However, you can change parabolic and s-curve profile jogs only to zero speed, and then only if the axis is not decelerating (Decel_status = 0). Unlike a move, changing the speed of a jog to zero terminates the jog and Jog_status = 0 (Not Jogging).

Analog Offset



The Analog Offset block lets the motion controller use an analog input to generate a scaled position offset and add that offset to the command position of the selected servo axis. By offsetting the controller's servo loop in response to the analog input signal, the servo loop can directly regulate pressure, force, tension, and more. To use analog inputs, you need to connect an analog sensor or transducer using a Flex I/O analog input module (p/n 1794-IE8 or 1794-IE4XOE2) to the controller.

The Analog Offset block resides on the Advanced Motion Palette.

ATTENTION: For maximum safety, be sure to set soft and hard limits before executing the Analog Offset block.

Immediately after the Analog Offset block executes, the axis moves from the actual input starting value to find the entered analog setpoint. The direction of the initial move depends upon the sign of the initial input value and the polarity you select in the Analog Offset block.

Calculating the Analog Input Scalar

The analog input scalar adjusts the system response to the analog input as reflected in the motion of the axis and may initially be estimated as follows:

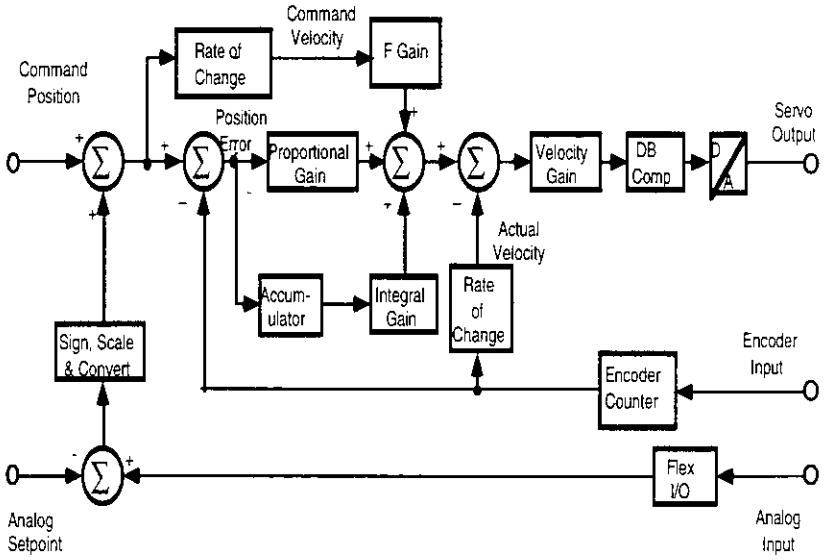
$$\text{Analog Input Scalar} = \frac{[\text{Maximum Voltage or Current Range (V or A) of Analog Input Sensor}] \times [\text{Servo Update Rate (Hz)}]}{[\text{Maximum Velocity of Axis (Units / Sec)}]}$$

ATTENTION: A too small analog input scalar value can cause the axis to oscillate, due to the slower Flex I/O module response time.

Note: This analog input scalar formula is only an estimate, and can cause a softer response than required. You can decrease the analog input scalar value for a sharper response, or increase it for a softer response. You need to evaluate each analog offset application individually, then adjust and test the scalar to provide a proper system response within a tolerable safety margin.

Analog Offset Control Loop Operation

When you use the analog offset function, the controller reads the value from the Flex I/O analog input module, adjusts the polarity if requested, and computes the difference between it and the specified analog setpoint. This value is then divided by the Analog Input Scalar, multiplied by the K constant specified for that particular axis (the Feedback page of the Conversion Constant set in the Configure Axis Use dialog box), and provides an analog error value in encoder counts. The servo loop in the motion controller seeks to null (reduce to zero) this error between the setpoint and the actual analog input. This is done using the encoder counts in commanding motion that is required for maintaining the analog setpoint. When you operate in Analog Offset mode, the motor transducer information and its feedback loop implementation remain unchanged. The figure below shows the servo loop after an Analog Offset block is executed.



Two Analog Offset axes system variables are available in GML Commander:

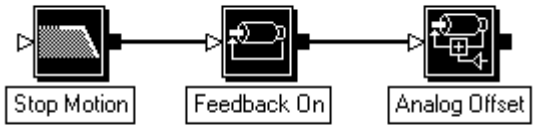
- the Analog_Offset_setpoint

- the Analog_Offset_error.

The Analog_Offset_setpoint axes system variable is the setpoint specified in the analog offset function block. The Analog_Offset_error is a system variable that contains the calculated difference between the current Flex I/O analog input and the Analog_Offset_setpoint. These variables are in the same units as those of the predefined Flex I/O analog input module.

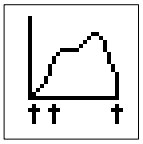
Programming an Analog Offset Function

To enable an Analog Offset Function, use the following GML Commander blocks after you have configured the Flex I/O Analog Input Module.



Before you can activate the Analog Offset function, the feedback for the configured axis must be turned on.

Configure Cam



Use the Configure Cam block to set up the necessary conditions for executing a time-lock or position-lock cam, or to set up a pending position-lock cam to blend one position-lock cam profile into another. Typically, this block immediately precedes a Time Lock Cam block or a Position Lock Cam block

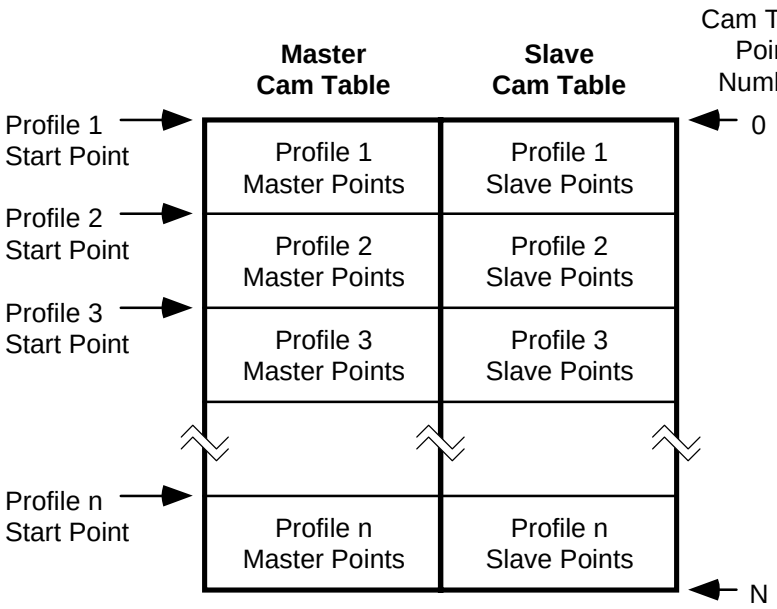
Also, use the Configure CAM block's Auto-Correction dialog box to continuously re-synchronize position-lock cam master and slave axes to registration marks. See *Auto-Correction* for details.

The Configure Cam function block resides on the CAM Palette.

Selecting Auto-Correction displays the Auto-Correction page for this dialog box. See *Auto-Correction* in this section for instructions on the Auto-Correction function.

Cam Start and End Points

The motion controllers contain two cam tables that can be used to store many individual time-lock and/or position-lock cam profiles. The master cam table stores time values (for time-lock cams) or master axis positions (for position-lock cams), and the slave cam table stores the corresponding slave axis positions, as shown below.



N = 1999 for iCODE V2.4 and earlier;
12999 for iCODE V3.0 and later.

The cam tables in the motion controller can store any number of individual profiles of any length as long as the total number of points for all profiles does not exceed the capacity of the table. For iCODE versions 3.0 or later (selected in the General page of the Configure Control Options dialog box), each cam table can contain up to 13,000 points, addressed 0 through 12999. For iCODE versions 2.4 or earlier, each cam table can contain up to 2,000 points addressed 0 through 1999.

Each point in a cam profile consists of two values—one for the time or master axis position (in the master cam table), and one for the slave axis position (at the corresponding location in the slave cam table). See the *Build Table* section of the *Calculation Blocks* chapter for more information on constructing cam tables. The start point for any individual cam profile is the cam table point number (0 – 1999 or 0 – 12999) of the first point in the appropriate profile. Likewise, the end point is the cam table point number of the last point in that profile.

For example, in the previous figure, assume cam profiles 1, 2, and 3 each consist of 100 points. Thus, profile 1 uses cam table points 0 – 99, profile 2 uses points 100 – 199, and profile 3 uses points 200 – 299. To use profile 2, enter 100 and 199 for the start and end points respectively.

Configuring Time Lock Cams

Selecting Once in the *Perform Profile* field causes the selected cam profile to execute only one time, when initiated by a subsequent Time Lock Cam block. See *Time Lock Cam* in this chapter for more information on time-lock cams.

Selecting Continuously in the *Perform Profile* field causes the selected cam profile to execute indefinitely, when initiated by a subsequent Time Lock Cam block. When program execution reaches the selected profile's end point, the profile repeats beginning at the specified start point. To generate smooth continuous motion using this technique, however, take care in designing the cam profile to ensure that there are no position, velocity, or acceleration discontinuities between the start point and end point of the profile.

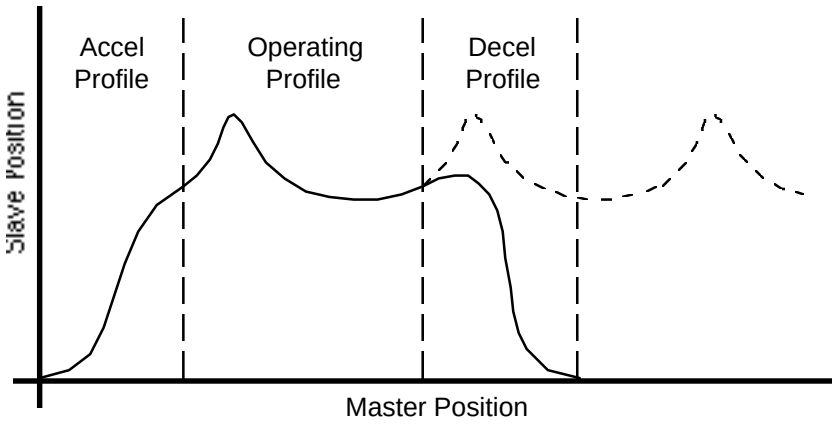
Configuring Position Lock Cams

Selecting Once in the *Perform Profile* field causes the selected cam profile to execute only once, when initiated by a subsequent Position Lock Cam block. This means that cam motion of the slave axis starts when the master axis moves into the range defined by the start and end points of the profile. When the master axis moves outside the range of the profile, cam motion on the slave axis stops; it resumes when and if the master moves back into the profile range specified by the start and end points.

Selecting Continuously in the *Perform Profile* field causes the selected cam profile to execute indefinitely, when initiated by a subsequent Position Lock Cam block. With continuous operation, the profile master and slave positions are unwound when the position of the master axis moves outside the profile range, causing the cam profile to repeat indefinitely. This feature is particularly useful in rotary applications where it is necessary that the position-lock cam run continuously in a rotary or reciprocating fashion. To generate smooth continuous motion using this technique, however, take care to design the cam profile to ensure that there are no position, velocity, or acceleration discontinuities between the start and end points of the profile.

Configuring Pending Position Lock Cams

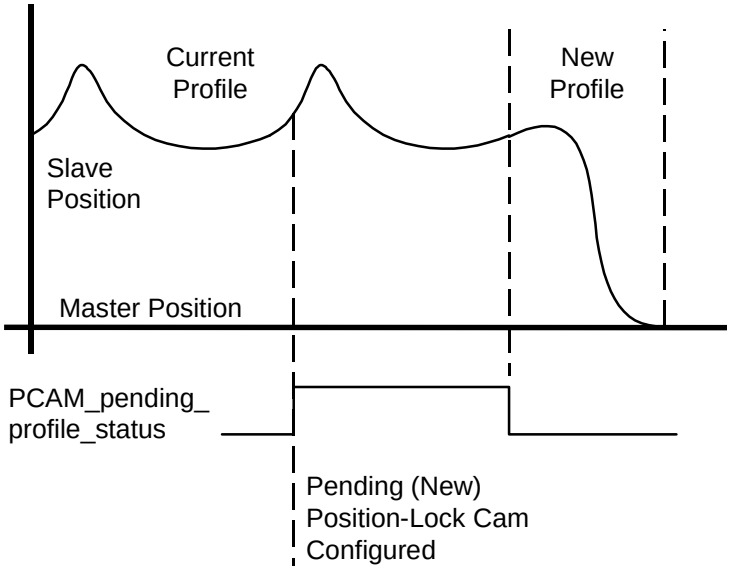
Use a Pending Position Lock Cam to seamlessly blend two position-lock cam profiles together without stopping either axis. This is useful in such applications as high-speed packaging, when a slave axis must lock onto a moving master axis, and accelerate to the proper speed using a specific profile. When this acceleration profile finishes, your program must smoothly blend it into the operating profile, which typically executes continuously. To stop the slave axis, your program must smoothly blend the operating profile into a deceleration profile, so that the axis stops at a known location, as shown below.



By configuring the next cam profile as a pending position-lock cam while the current profile still executes, you set the appropriate parameters ahead of time. This makes the transition from the current profile to the pending profile seamless—synchronization between the master and slave axes is maintained. To ensure smooth motion across the transition, however, be sure to carefully design your cam profiles so that no position, velocity, or acceleration discontinuities exist between the end of the current profile and the start of the new one.

Unlike regular position-lock cams, initiating the new (pending) cam profile requires no subsequent Position Lock Cam block. Once a pending position-lock cam has been configured, the new profile takes effect automatically (and becomes the current profile) the next time the master axis passes through either the start or end point of the current profile. If the current cam is configured to execute continuously, the new profile initiates upon the completion of the current pass through the current profile. The motion controller keeps track of the master and slave axis positions, relative to the first profile at the time of the change, and uses this information to maintain synchronization between the profiles.

After a pending position-lock cam has been configured, the PCAM_pending_profile_status variable for the selected slave axis is 1 (true). When the new (pending) profile is initiated and becomes the current profile, PCAM_pending_profile_status = 0 as shown below.

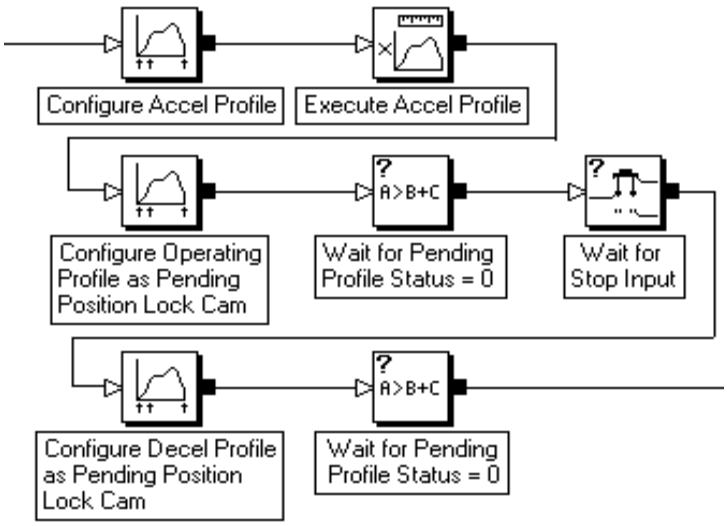


See *System Variable* in the *Expression Builder* chapter of this manual for more information on variables.

Selecting Once in the *Perform Profile* field causes the new (pending) cam profile to initiate and execute only once, when the master axis passes through either the start or end point of the current profile. When the master axis moves outside the range of the new profile, cam motion on the slave axis stops; it resumes when and if the master moves back into the profile range specified by the start and end points.

Selecting Continuously in the *Perform Profile* field causes the new (pending) cam profile to initiate and execute indefinitely, when the master axis passes through either the start or end points of the current profile. With continuous operation, the new profile master and slave positions are unwound when the master axis moves outside the profile range, causing the cam profile to repeat indefinitely. To generate smooth continuous motion using this technique, however, take care in designing the cam profile to ensure that there are no position, velocity, or acceleration discontinuities between the start and end points of the profile.

The GML Commander diagram below implements the three-part position-lock cam profile shown earlier.



The acceleration profile is first configured and then executed. The operating profile is then immediately configured as a pending position-lock cam which takes effect as soon as the acceleration profile has completed. The operating profile is executed continuously until the stop input is activated, at which time the deceleration profile is configured as another pending position-lock cam. The deceleration profile is executed at the conclusion of the current pass of the operating profile to bring the axis smoothly to a stop.

Auto Correction (Configure Cam)

Use Auto-Correction, in conjunction with Auto Registration, to continuously re-synchronize position-lock cam master and slave axes to registration marks. Continuous re-synchronization is necessary when an axis slips, or if the material, upon which the registration marks are printed, is not consistent. To enable auto-correction, you must first enable auto registration for the registering axis with a Watch Control block (set to Arm Registration). See the *Watch Control* section of the *I/O and Event Blocks* chapter for more information on enabling auto registration.

Selecting Auto-Correction**,** in the Configure Cam dialog box enables the Auto-Correction dialog box, for Position-Lock cams or Pending Position-Lock cams (but not Time-Lock cams).

The Auto-Correction dialog box presents a varying array of fields, depending upon your choice Auto-Correction Type in the Auto-Correction page.

ATTENTION: Do not use auto-correction unless you have previously enabled auto-registration for the registering axis in an Arm Registration block.

When a Configure Cam block executes with one of the auto-correction types selected, the auto-correction status variable for the selected slave axis is set (PCAM_auto_correction_status = 1).

If auto-correction is subsequently disabled when another Configure Cam block is executed, the auto-correction status variable for the selected slave axis is reset (PCAM_auto_correction_status = 0).

See the *System Variables* chapter in this manual for more information on this variable.

Phase Shift and Incremental Auto-Correction

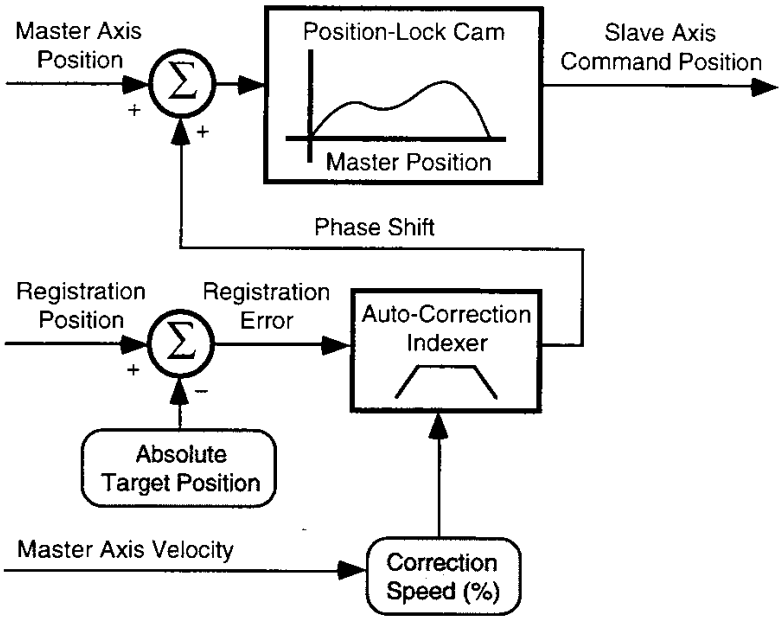
Select Phase to perform auto correction *only* by phase shifting the slave axis relative to the master axis. This is the default selection. All four auto correction types (relative, absolute, relative ratioed, and absolute ratioed) use phase shift auto correction, whether or not you select Phase (even if you select Incremental).

Select Incremental to perform a second level of auto correction by incrementing (adding or subtracting) a constant value to the slave axis values set in the cam table. Selecting Incremental is much the same as executing an incremental Move Axis block for the slave axis while the position lock cam is operating.

See *Moving While Camming* in the *Position Lock Cam* section of this chapter for more information on the effects of incremental moves and phase shift moves of the slave axis cam values.

Absolute Auto-Correction

Use absolute auto-correction to automatically correct for registration error via a phase shift move of the slave axis upon the occurrence of every registration event, as shown in the following figure.



When a good registration event occurs on the registering axis, the PCAM_good_registration_count axis variable is incremented and the absolute position is subtracted from the appropriate registration position to determine the registration error. This calculated registration error is then used to phase shift the slave axis via a dedicated auto-correction indexer. See *Phase Shift Moves* in *Move Axis* in this section for more information on phase shift moves. See the *Tolerance* section for the definition of a good registration event.

If the auto-correction move finishes before the next good registration event occurs, the registration error is completely corrected. If the current auto-correction move does not finish by the time the next good registration event occurs, the new registration error overwrites the remaining distance to go of the auto-correction indexer. In this case, then, the original registration error is not completely corrected.

When absolute auto-correction is enabled:

- PCAM_auto_correction_status = 1

- the PCAM_registration_error variable displays the current registration error value
- the PCAM_average_registration_error variable displays the average registration error over the last ten registrations
- the PCAM_auto_correction_to_go variable displays the current remaining distance to go of the auto-correction indexer
- the PCAM_good_registration_count variable displays the current number of good registration events
- the PCAM_bad_registration_count variable displays the current number of consecutive out-of-tolerance registration events
- the PCAM_missing_registration_count variable for the slave axis displays the current number of consecutive missed registration counts

See the *System Variables* chapter of this manual for more information on these variables.

Absolute (Target) Position

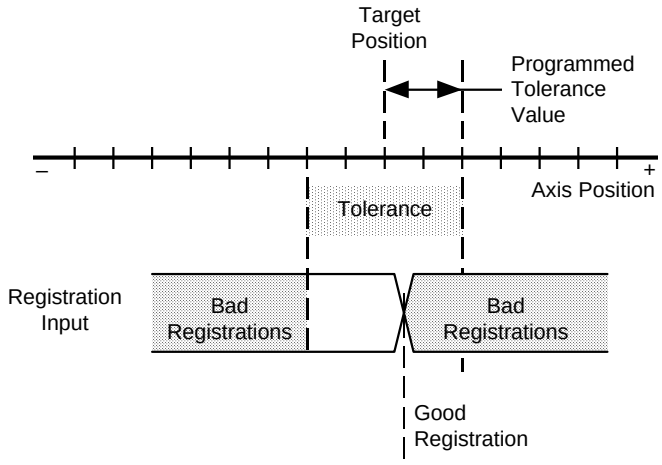
Enter a value or expression equal to the expected or desired absolute position of the registration event on the registering axis in the position units of the Master Axis in the *Absolute Position* field. This is the value that is subtracted from the measured registration position whenever a good registration event occurs to determine the phase shift correction for the slave axis.

Tolerance

A tolerance is a range on either side of the registration (target) position. Setting a tolerance limits error correction to only those errors that fall within this range, and prevents the motion controller from trying to correct unusually large errors. Large errors mean that serious problems exist, either with the hardware or the software running it. These large errors go well beyond mere axis slippage, or inconsistent material errors that auto-correction can remedy.

ATTENTION: Using auto-correction to correct large errors can cause harsh motion and result in damage to both equipment and materials.

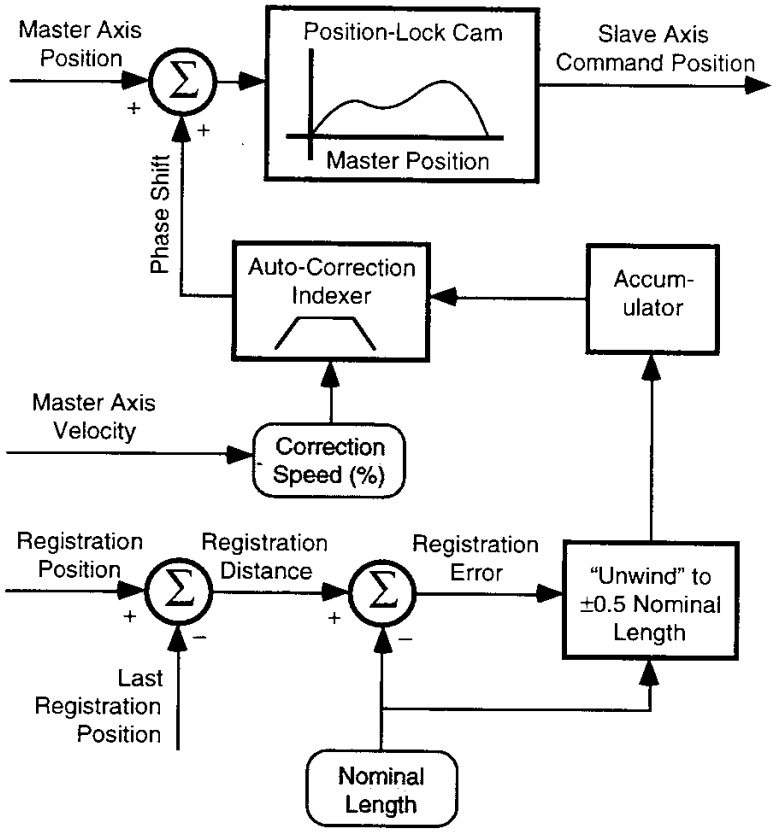
To allow rejecting spurious registration events, specify a tolerance around the target position by selecting Tolerance, and enter a value or expression equal to the desired target position tolerance in the data entry box. The specified tolerance is interpreted as a ± quantity around the Absolute Target Position and used to distinguish good registration events from bad ones, as shown below.



As shown above, the first registration event that occurs within the specified tolerance is considered the good registration. All registration events that occur after a good registration event and before the beginning of the next tolerance area are considered bad, even if they occur within the current tolerance. If Tolerance is not selected, all registration events on the registering axis are considered good.

Relative Auto-Correction

Relative Auto-Correction automatically corrects for error trends in the distance between registration events via a phase shift move of the slave axis as shown below.



When a good registration event occurs on the registering axis, the PCAM_good_registration_count axis variable is incremented and the previous registration position is subtracted from the appropriate registration position to determine the distance between this registration event and the last one. The specified nominal length is then subtracted from this registration distance to determine the registration error. This registration error is then unwound to ±1/2 the specified nominal length, accumulated, and used to phase shift the slave axis via a dedicated auto-correction indexer to correct for error trends in the registration distance. See *Phase Shift Moves* in the *Move Axis* section of this chapter for more information on phase shift moves.

The maximum magnitude of the auto-correction move for each cycle is limited to ±1/2 the specified nominal length by adding or subtracting the nominal length value to/from the registration error if it exceeds ±1/2 the specified nominal length. This unwinding of the registration error means that the slave axis always phase shifts to the closest synchronization point within the current product cycle.

In addition, unlike absolute auto-registration, if the relative auto-correction move does not complete before the next good registration event occurs, the new registration error is added to the remaining distance to go of the auto-correction indexer. By accumulating the registration error in this manner, the registration events can occur much more frequently than with absolute auto-correction, where the full registration error must be corrected within the current product cycle. This causes relative auto-correction to correct for trends in the registration error rather than the full registration error.

When relative auto-correction is enabled:

- PCAM_auto_correction_status = 1
- the current registration error is available in the PCAM_registration_error variable
- the average registration error over the last ten registrations is available in the PCAM_average_ registration_error variable

- the current remaining distance to go of the auto-correction indexer is available in the PCAM_auto_ correction_to_go variable
- the current number of good registration events is available in the PCAM_good_registration_count variable
- the current number of consecutive out-of-tolerance registration events is available in the PCAM_bad_registration_count variable
- and the current number of consecutive missed registration events is available in the PCAM_missing_registration_count variable for the slave axis

See the *System Variables* chapter of this manual for more information on these variables.

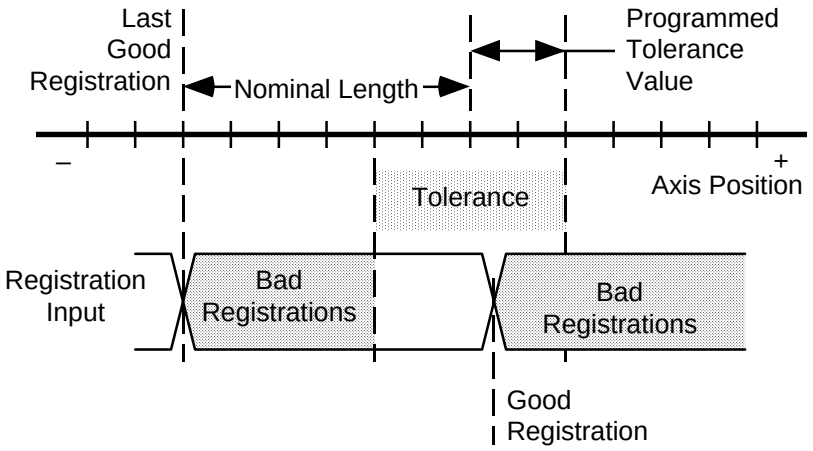
Nominal Length

In the *Nominal Length* field, enter a value or expression equal to the expected or desired distance between registration events on the registering axis, in the position units of the selected master axis. This is the value that is subtracted from the measured registration distance whenever a good registration event occurs and then used to determine the registration error.

Tolerance

A tolerance is a range on either side of the nominal length. Setting a tolerance limits error correction to only those errors that fall within this range, and prevents the motion controller from trying to correct unusually large errors. Large errors usually mean that serious problems exist, either with the hardware or the software running it. These large errors go well beyond mere axis slippage, or inconsistent material errors that auto-correction can remedy. Using auto-correction to correct these large errors can cause harsh motion and result in damage to both equipment and materials.

To allow rejecting spurious registration events, a tolerance may be specified around the nominal length by selecting Tolerance, and entering a value or expression equal to the desired nominal length tolerance in the data entry box. The specified tolerance is interpreted as a ± quantity around the specified nominal length and used to distinguish good registration events from bad ones, as shown below.



As shown above, the first registration event that occurs within the specified tolerance is considered the good registration. All registration events that occur after a good registration event and before the beginning of the next tolerance area are considered bad, even if they occur within the current tolerance. If Tolerance is not selected, all registration events on the registering axis are considered good.

Last Registration

When you first enable relative auto-correction, it is often necessary to have a default value for the last registration position (see the figure earlier in this section) to ensure a reasonable registration distance calculation upon the occurrence of the first registration event.

To specify such a default last registration position, select Last Registration, and enter the desired value or expression, in the position units of the master axis, into the corresponding field.

In most applications, the last registration position value should be equal to the ideal registration position within the product cycle. If you do not select Last Registration, the value of the last registration position upon the occurrence of the first good registration event is undefined, in which case the controller does not perform a phase correction. Instead, it uses the position of this first registration event as the last registration and begins phase correcting on the occurrence of the next (second) registration event.

Absolute Ratioed Auto-Correction

Absolute ratioed auto-correction (like absolute auto-correction) automatically corrects for registration error, using a phase shift move of the slave axis, upon the occurrence of every registration event. However, with absolute ratioed auto-correction, the registration error is calculated in registering axis (not master axis) position units and converted to master axis units using the ratio (in feedback counts) of the master axis cycle and registering axis cycle. This allows using the hardware registration input of the registering axis rather than the soft registration position of the master axis when the hardware registration input of the master axis is already used.

When a good registration event occurs on the registering axis, the PCAM_good_registration_count axis variable is incremented. The absolute position (targeted position) is subtracted from the registration position of the registering axis to determine the registration error in the position units of the registering axis. This value is converted to the registration error in the position units of the master axis by multiplying it by the ratio of the master axis cycle to the registering axis cycle as shown below.

$$\text{Reg. Error [Master Axis]} = \text{Reg. Error [Reg. Axis]} \times \frac{\text{Master Cycle}}{\text{Registering Cycle}}$$

The calculated registration error in the position units of the master axis is then used to phase shift the slave axis via a dedicated auto-correction indexer as shown earlier for absolute auto-correction.

See *Phase Shift Moves* in the *Move Axis* section of this chapter for more information on phase shift moves. See *Absolute Auto-Correction* in this section for more information on the operation of absolute ratioed auto-correction.

ATTENTION: Be sure to enter the master axis cycle and registering axis cycle values in feedback counts, not position units.

The corresponding parameters for absolute ratioed auto-correction are the same as those for absolute auto-correction with the exception that the absolute position, tolerance, no correction start point, and no correction end point values are entered in the position units of the registering axis, not the master axis. In addition, enter the number of feedback counts for one product cycle on the master axis and for one product cycle on the registering axis in the *Master Axis Cycle* and *Registering Axis Cycle* fields respectively. These values must be in feedback counts, not position units.

Relative Ratioed Auto-Correction

Relative ratioed auto-correction (like relative auto-correction) automatically corrects for error trends in the distance between registration events, using a phase shift move of the slave axis. However, with relative ratioed auto-correction, the registration distance and registration error are calculated in the position units of the registering axis (not the master axis) and converted to master axis units using the ratio of the master axis cycle and the registering axis cycle. This allows using the hardware registration input of the registering axis rather than the soft registration position of the master axis when the hardware registration input of the master axis is already used.

When a good registration event occurs on the registering axis, the PCAM_good_registration_count variable for the axis is incremented. The previous registration position is subtracted from the registration position of the registering axis to determine the distance between this registration event and the last in the position units of the registering axis. The nominal length is then subtracted from this registration distance to determine the registration error. This calculated registration error is then unwound to $\pm 1/2$ the nominal length and accumulated. Finally, the registration error in the position units of the registering axis is converted to the registration error in the position units of the master axis by multiplying it by the ratio of the master axis cycle to the registration axis cycle as shown below.

$$\text{Reg. Error [Master Axis]} = \text{Reg. Error [Reg. Axis]} \times \frac{\text{Master Cycle}}{\text{Registering Cycle}}$$

The calculated registration error in the position units of the master axis is then used to phase shift the slave axis via a dedicated auto-correction indexer to correct for error trends in the registration distance as shown earlier for relative auto-correction.

See *Phase Shift Moves* in the *Move Axis* section of this chapter for more information on phase shift moves. See *Relative Auto-Correction* in this section for more information on the operation of relative ratioed auto-correction.

ATTENTION: Be sure to enter the master axis cycle and registration axis cycle values in feedback counts, not position units.

The corresponding parameters for relative ratioed auto-correction are the same as those for relative auto-correction with the exception that the nominal length, tolerance, no correction start point, no correction end point, and last registration values are entered in the position units of the registration axis, not the master axis. In addition, enter the number of feedback counts for one product cycle on the master axis and for one product cycle on the registering axis in the *Master Axis Cycle* and *Registering Axis Cycle* fields respectively. These values must be in feedback counts, not position units.

Master Axis

You can select a master axis only for a subsequent Position-Lock cam block. Use a Pending Position Lock cam block to re-configures cam settings—other than the Master Axis setting, which remains unchanged—that were set in a previous Configure CAM block.

Select the same master axis selected in the subsequent Position Lock Cam block (which you use to initiate the cam) as the master axis. If the master axis specified for auto-correction in the Configure Cam block is not the same as the master axis specified for this slave axis in the appropriate Position Lock Cam block, an improper or inaccurate registration error will be calculated.

Registering Axis

Select the axis for which registration events initiate calculation of the registration error as the registering axis.

Hard Registration

The Registration_Position variable of the registering axis is used as the registration position for auto-correction if either of the following is true:

- the registering axis and the master axis are the same.
- you selected absolute ratioed or relative ratioed auto-correction type.

This hard registration position is the most accurate because it is latched in hardware within 1 µs of the transition of the corresponding registration input.

Soft Registration

The Soft_Reg_Pos_Master Axis_Registering Axis variable is used as the registration position for auto-correction if you selected either absolute or relative as the auto-correction type and the registering axis is not the same as the master axis.

For example, if you select AXIS0 as the master axis, and AXIS1 as the registration axis, the Soft_Reg_Pos_AXIS0_AXIS1 variable is used as the registration position for auto-correction.

Soft registration is not as accurate as hard registration, because the soft registration variables are sampled at the servo update rate rather than directly in hardware.

See the *System Variables* chapter of this manual for more information on the Registration_Position and Soft_Reg_Pos… variables.

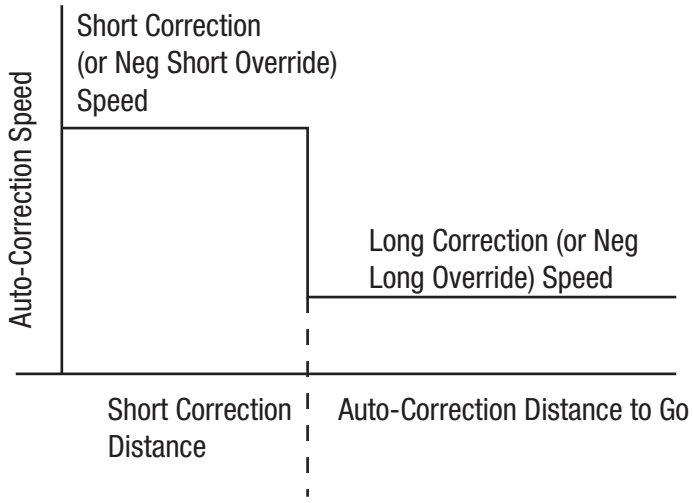
Auto-Correction Speed

The speed that the auto-correction indexer uses to phase shift the slave axis is proportional to the current speed of the Master Axis. Therefore, auto-correction occurs faster when the master is moving faster, and slower when the master is moving slower.

Generally, if the master axis is not moving at all, no auto-correction of the slave axis occurs. (See: Starting P-camming or Gearing in Using the Encoder Input Filter for an exception to this general rule.)

If the selected slave axis is cammed to the actual position of the master in the subsequent Position Lock Cam block, auto-correction speed is proportional to the actual instantaneous velocity–not the averaged velocity–of the master axis. Similarly, if the slave axis is cammed to the command position of the master axis, auto-correction speed is proportional to the command velocity of the master axis.

The speed used by the auto-correction indexer depends on the magnitude of the auto-correction distance-to-go, as shown below.



Auto-correction speed is set to:

- **Long Correction (or Neg Long Override) Speed** if the magnitude of the current auto-correction distance-to-go is greater than the specified Short Correction Distance,
- **Short Correction (or Neg Short Override) Speed** percentage of the current velocity of the Master Axis, if not.

The **Neg Short Override** and **Neg Long Override** settings let you separately set speeds for negative direction auto-corrections.

A long correction move starts out at the long correction (or negative long override) speed. When the auto-correction distance-to-go becomes less than or equal to the specified short correction distance, the auto-correction indexer accelerates or decelerates to the specified short correction (or negative short override) speed. Acceleration (or deceleration) occurs at the slave axis' Maximum Acceleration (or Maximum Deceleration) rate for the remainder of the correction.

Note that the short correction (negative short override) speed, as represented in the above figure, is faster than the long correction (negative long override) speed. This optional relationship is typical: short corrections can be made quickly, while larger corrections are usually less to avoid exceeding speed or acceleration limitations of the slave axis.

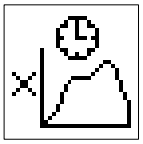
No-Correction Zone

In certain applications, you want to inhibit phase correction motion during a certain portion of the position-lock cam cycle. For example, it may be unwise to vary the speed of a cutting axis when cutting occurs. This could cause damage to either the material being cut, or the cutting implement.

With No Correction selected, the auto-correction indexer immediately stops the phase shift move of the slave axis when the controller reaches the no correction start point. The auto-correction indexer remains paused until the controller passes the no correction end point, at which point it resumes the phase shift move of the slave axis to complete the correction. The slave axis neither decelerates when entering, nor accelerates when leaving the no-correction zone. With large registration errors and/or high correction speeds, this can cause a position error fault on the slave axis. If No Correction had not been selected, the auto-correction phase shift move would finish without interruption. (As shown by the dotted line in the correction figure in Auto-Correction section of this chapter.)

ATTENTION: The slave axis neither decelerates, when entering, nor accelerates, when leaving, the no-correction zone. With large registration errors and/or high correction speeds, this can cause a position error fault on the slave axis.

Time Lock CAM



The Time Lock Cam block executes a time-lock cam profile created by a previous Configure Cam block, on the selected servo or imaginary slave axis, in the selected direction. Time-lock cams allow the execution of motion profiles other than trapezoidal, S-curve, or parabolic. No maximum velocity, acceleration, or deceleration limits are used for time-lock cams. The speed, acceleration, and deceleration of the slave axis are completely determined by the profile stored in the cam tables.

The Time Lock Cam block resides on the Cam Palette.

While a time-lock cam is in progress:

- TCAM_status = 1 (true)
- Axis_status = 3 (Executing Time-lock Cam)
- Lock_status = 0 (Unlocked)

When the cam is done:

- TCAM_status = 0 (false)
- Axis_status = 1 (Unlocked)

When the cam is done and the position error of the axis is less than the position lock tolerance:

- Axis_status = 0 (Locked)
- Lock_status = 1 (Locked)

See the *Axis Locked and Axis Done Conditions* chapter of this manual for a complete discussion of the difference between these conditions. See the *System Variables* chapter of this manual for more information on variables.

ATTENTION: Time lock cams neither recognize nor use the maximum speed, acceleration, or deceleration values set in the Dynamics page of the Configure Axis Use dialog box, or in a Motion Settings function block.

Time-lock cams are fully interpolated. If the current time does not correspond exactly with a point in the specified profile, the motion controller uses linear interpolation to calculate the (slave) axis position between the two closest points. This provides for the smoothest possible motion. Also, when a Gear Axes block is subsequently used on the axis, the time-lock cam remains active. This lets you superimpose time-lock cam motions on electronic gearing motion, thereby creating complex synchronization.

Selecting a Time Lock Cam Direction

Positive Direction

When you select positive in the direction field, the incremental distance values computed from the slave profile points are added to the command position of the axis. This is the most common operation, since axis motion is in the same direction as implied by the slave point values. That is, consecutive increasing profile values result in axis motion in the positive direction and vice-versa.

Negative Direction

When you select negative in the direction field, the incremental distance values computed from the slave profile points are subtracted from the command position of the axis. With this type of operation, axis motion is in the opposite direction from that implied by the slave point values. That is, consecutive increasing profile values result in axis motion in the negative direction and vice-versa.

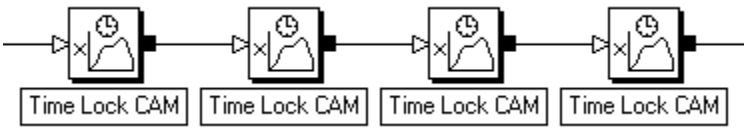
Merge from Jog

Select Merge from Jog to let a previously programmed Jog Axis block smoothly ramp the axis to the proper speed before the Time Lock Cam block is executed. This is useful when the cam must be run continuously, but the specified profile has a non-zero velocity at the start/end point.

For the smoothest transition between the jog and the time-lock cam, make sure that the speed entered in the previous Jog Axis block equals the speed implied for the axis at the start/end point of the specified profile. See *Configure Cam* in this chapter for information on specifying the desired cam profile.

Synchronizing Time Lock Cams on Multiple Axes

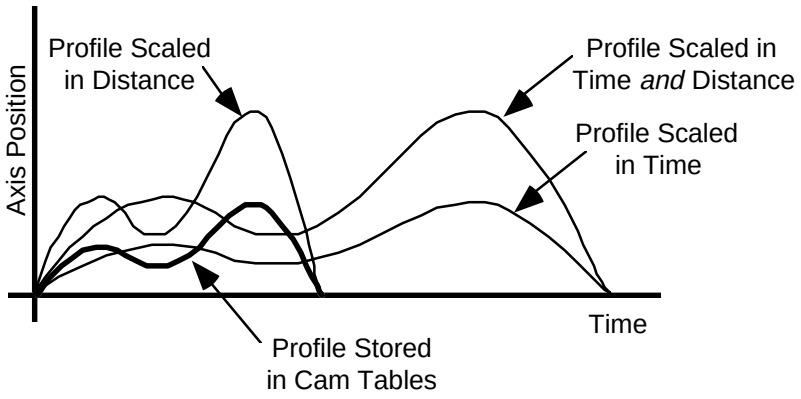
When you select Synchronize with next TCam, the current Time Lock Cam block executes simultaneously with the following Time Lock Cam block. This lets you simultaneously begin cams on multiple axes, as shown below.



When you select Synchronize with next TCam, the block immediately following the Time Lock Cam block must be another Time Lock Cam block GML Commander does not let you connect the two blocks. In the example above, select Synchronize with next TCam in the first three Time Lock Cam blocks—but not in the last block—to start all four axes simultaneously.

Scaling Time Lock Cams

You can scale a time-lock cam profile in both time and distance when it is executed. Selecting Scale Profile uses the stored profile for only the shape of the motion, and scales the profile definition by the total time and distance over which the profile is to be executed, as shown below.

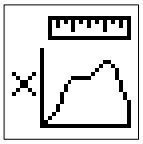


To scale a:	Do this:
Time-lock cam profile	1. Select Scale Profile. 2. Enter the desired total time in seconds and the total distance (in position units of the axis) over which the profile is to be executed.
Profile only in time	Enter a total distance equal to the distance moved by the axis when the profile is executed without scaling (as implied by the defined profile in the slave cam table).
Profile only in distance	Enter a total time equal to the motion time when the profile is executed without scaling (as implied by the defined profile in the cam time table).

ATTENTION: When you use scaling to decrease the total time, or increase the total distance of a time-lock cam, the required velocities and accelerations of the profile increase. This causes a motion fault if the increased velocities and accelerations exceed the drive system's capabilities.

Increasing the total time of a cam profile decreases the velocities and accelerations of the profile, while increasing the total distance increases the velocities and accelerations of the profile. To maintain the velocities and accelerations of the scaled profile approximately equal to those of the unscaled profile, total time and total distance should be scaled in proportion to one another. For example, if you double the total distance of a profile by scaling, the total time should also be doubled to maintain approximately equal velocities and accelerations during execution of the scaled time-lock cam.

Position Lock CAM



Use the Position Lock Cam block to execute a position-lock cam profile set up by a previous Configure Cam block on the selected servo or imaginary slave axis, in the selected direction, using the selected master axis.

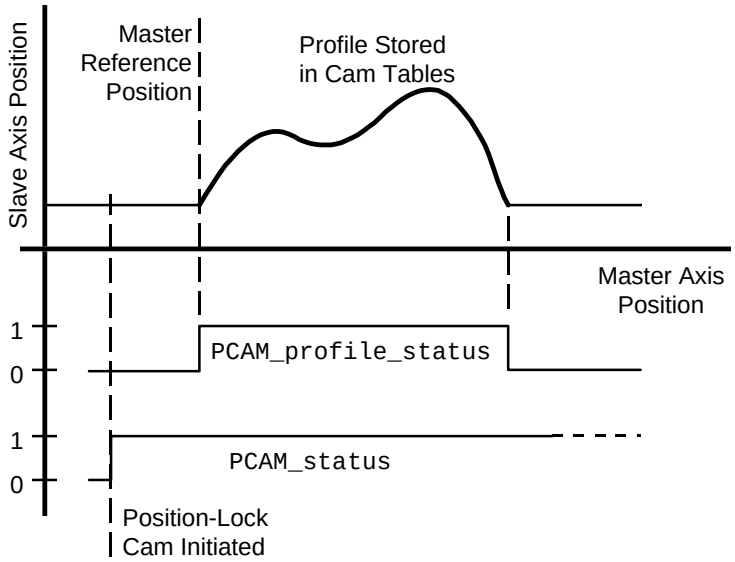
Position-lock cams provide the capability of implementing non-linear electronic gearing relationships between two axes. No maximum velocity, acceleration, or deceleration limits are used. The speed, acceleration, and deceleration of the slave axis is completely determined by the motion of the master axis and the profile stored in the cam tables.

The Position Lock Cam block resides on the CAM Palette.

Position lock cams neither recognize nor use the maximum speed, acceleration, or deceleration values entered in the Dynamics page of the Configure Axis Use dialog box, or in a Motion Settings function block.

.

When:	Then:
A position-lock cam is initiated and is active for the axis	PCAM_status = 1 (true).
The cam has been initiated and the specified master axis is within the range defined by the master cam table	PCAM_profile_status = 1.



When PCAM_status = 1, Axis_status ≤ 5 if no faults are active on the axis.

If the cam is disabled or all motion on the slave axis is stopped, PCAM_status = 0 (false).

See the *System Variables* chapter of this manual for more information on these variables. Note that the cam is not automatically disabled when the master axis moves out of the range defined by the master cam table. See *Disable PCAM* and *Stop Motion* sections for more information on disabling and stopping position-lock cams.

Position-lock cams are fully interpolated. If the current master axis position does not correspond exactly with a point in the specified profile, the motion controller uses linear interpolation to calculate the slave axis position between the two closest points. This provides for the smoothest possible motion. Also, when a Jog Axis or Move Axis block is subsequently used on the axis, the position-lock cam remains active. This lets you superimpose time-lock cam motions on electronic gearing motion, thereby creating complex synchronization.

See the *Axis Locked and Axis Done Conditions* chapter of this manual for a complete discussion of the operation of these status conditions when a move, jog, or time-lock cam is superimposed on position-lock cam motion.

Virtual Master Axes

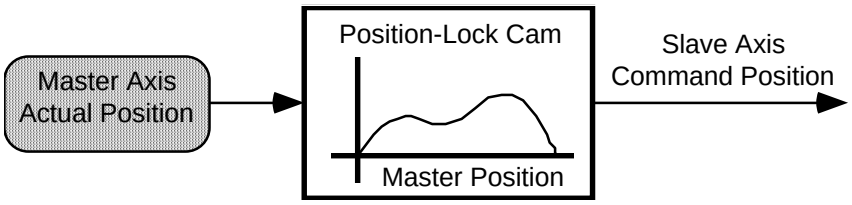
Because AxisLink virtual axes are functionally equivalent to physical Master Only axes, they can be used as the master axis for position-lock cams. However, only one of the two available virtual axes can be enabled at one time—enabling one virtual axis automatically disables the other virtual axis if it had previously been enabled. Thus, you must first enable an AxisLink virtual axis, with a previous Virtual Axis Control block, before it can be used as the master axis for a position-lock cam. See *Virtual Axis Control Block* in the *AxisLink Block* chapter for information on enabling virtual axes.

Position-Lock Cams and the Imaginary Axis

You can use the imaginary axis as either the master or the slave axis for position-lock cams. However, because the imaginary axis' output is its command position—it has no actual position—only Command is available for selection.

Slaving to the Actual Position

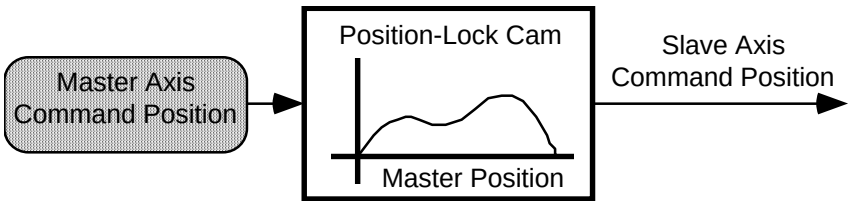
When you select Actual from the Slave to menu, the actual position of the master axis generates the position-lock cam motion on the slave axis, as shown below.



Actual position is the current position of a physical or virtual master axis as measured by its encoder or other feedback device. This is the usual selection—and the only selection when the master axis is a master only axis (physical or virtual)—because it is most often necessary to synchronize the actual positions of two axes.

Slaving to the Command Position

When you select Command from the Slave to menu, the command position of the master axis generates the position-lock cam motion on the slave axis, as shown below.



Command position (only available when the master axis is a servo or the imaginary axis) is the intended or commanded position of the master axis, as generated by any previous motion blocks. Because virtual axes are equivalent to physical master only axes, only actual position is available.

Because the command position does not incorporate any associated following error or external position disturbances, it is a more accurate and stable reference for camming. When camming to the command position of the master, the master axis must be commanded to move in order to cause any motion on the slave axis. See the *Installation and Setup* manual for your motion controller for more information on command position.

Selecting a Position-Lock Cam Direction

Camming in the Same Direction

When you select Same in the *Direction* field, the slave axis position values computed from the slave profile points are added to the command position of the slave axis. This is the most common operation, as the profile point values are used directly. That is, continuously increasing profile values results in axis motion in the positive direction, and vice-versa.

Camming in the Opposite Direction

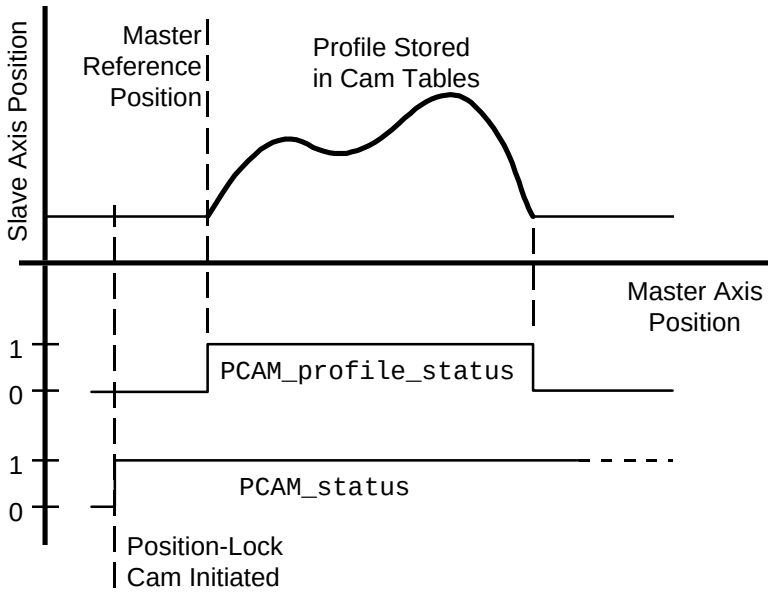
When you select Opposite from the Direction menu, the slave axis position values computed from the slave profile points are subtracted from the command position of the slave axis. Thus, axis motion is in the opposite direction from that implied by the slave point values. That is, continuously increasing profile values results in axis motion in the negative direction, and vice-versa.

Reversing the Camming Direction

When you select Reverse from the Direction menu, the current direction of the position-lock cam reverses itself, as if you had changed the setting from same to opposite or from opposite to same.

Master Reference Position

As shown below, when you execute the position-lock cam block, the cam is initiated on the specified slave axis and PCAM_status for the slave axis is set to 1. The position of the master axis is then monitored to determine when the master axis passes the specified master reference position.



When the absolute position of the master axis passes the specified master axis reference position and enters the range defined by the master cam table, PCAM_profile_status for the specified slave axis is set to 1 and slave axis motion is initiated according to the specified profile. From this point on, the incremental change in the master axis position alone determines the corresponding slave axis position from the defined profile points. This is important for applications where the master axis is a rotary axis, because the position-lock cam remains unaffected by the unwind process.

When the master axis moves out of the range defined by the master position table, PCAM_profile_status = 0, but the cam is not automatically disabled (PCAM_status = 1).

See *Disable PCAM* and *Stop Motion* in this section for more information on disabling and stopping position-lock cams. See the *System Variables* in this manual for more information on the PCAM_status and PCAM_profile_status variables.

The position-lock cam motion starts when the master axis position passes the specified master reference position. Once the cam starts, however, the master axis can change direction while in the defined profile range and the slave axis reverses accordingly. In addition, if the master axis position leaves the defined profile range, slave motion stops but restarts when the master re-enters the range, regardless of which direction the master is moving when it re-enters the defined profile range. See *Configure Cam* in this section for information on specifying the desired cam profile.

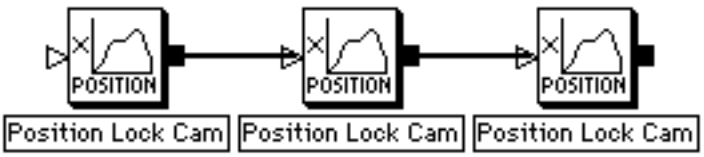
Unidirectional Position-Lock Cams

Uni-directional only position-lock cam operation provides an electronic slip clutch which prevents the slave axis from moving backward through the cam profile if the master backs up (moves backward). When you select uni-directional only, the slave axis tracks the master axis in the direction implied by the slave cam table points (i.e. increasing point values cause motion in the positive direction and vice versa) when the master moves in the direction implied by the master cam table points. If the master axis changes direction, the slave axis does not reverse direction, but stays where it was when the master reversed.

When the master axis again reverses (resuming motion in the direction implied by the master cam table points) the slave axis resumes its motion when the master reaches the position where it initially reversed. In this way, the slave axis maintains synchronization with the master while motion in the wrong direction is inhibited. This is especially useful where backward motion can cause physical damage to the machine.

Synchronizing Position-Lock Cams on Multiple Axes

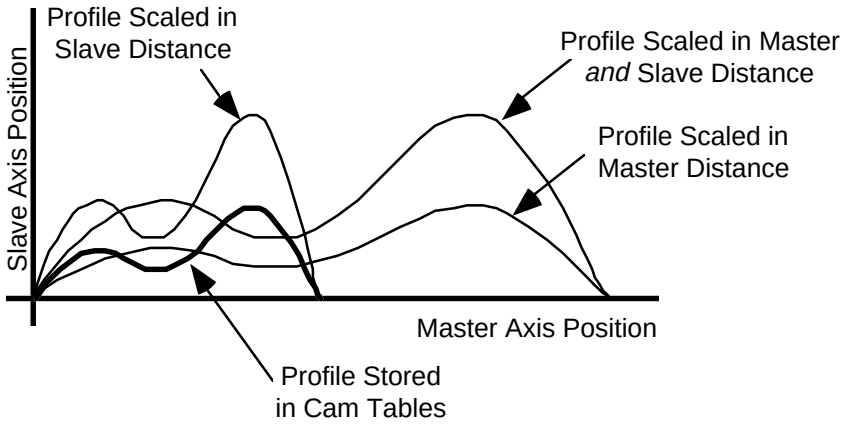
When you select Synchronize with next PCam, the current Position Lock Cam block executes simultaneously with the following Position Lock Cam block. This lets you simultaneously begin cams on multiple axes, as shown below.



When you select Synchronize with Next PCam, the block immediately following the position-lock cam block must be another Position Lock Cam block or GML Commander does not permit you to connect the two blocks. In the example above, select Synchronize with Next PCam in the first two Position Lock Cam blocks—but not in the last block—to start all three axes simultaneously.

Scaling Position-Lock Cams

You can scale a position-lock cam profile in terms of both master distance and slave distance when it is executed. Selecting Scale Profile uses the stored profile for only the shape of the motion, and scales the profile definition by the total master distance and slave distance over which the profile is to be executed, as shown below.



To scale a position-lock cam profile, select Scale Profile and enter the desired total master distance (in the position units of the master axis) and total slave distance (in position units of the slave axis) over which the profile is to be executed.

To scale a profile only in:	Do this:
Master distance	Enter a total slave distance equal to the distance moved by the slave axis when the profile is executed without scaling (as implied by the defined profile in the slave cam table).
Slave distance	Enter a total master distance equal to the distance moved by the master axis when the profile is executed without scaling (as implied by the defined profile in the master cam time table).

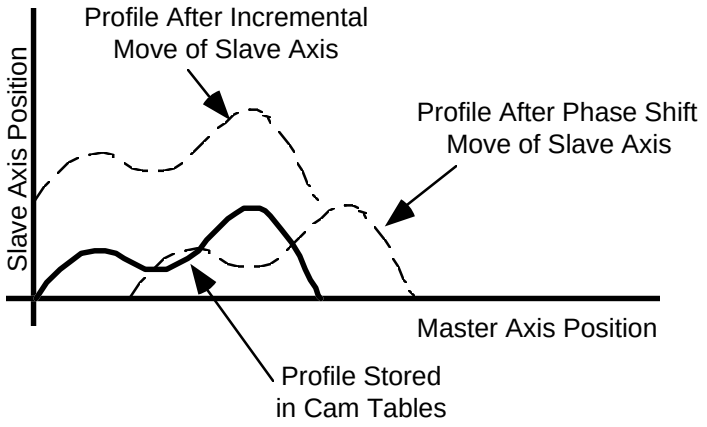
ATTENTION: Decreasing the total master distance or increasing the total slave distance of a position lock cam by scaling increases the required velocities and accelerations of the profile. This causes a motion fault if the capabilities of the drive system are exceeded.

Increasing the total master distance of a cam profile decreases the velocities and accelerations of the profile, while increasing the total slave distance increases the velocities and accelerations of the profile. To maintain the velocities and accelerations of the scaled profile approximately equal to those of the unscaled profile, total master distance and total slave distance should be scaled in proportion to one another. For example, if the total slave distance of a profile is doubled by scaling, the total master distance should also be doubled to maintain approximately equal velocities and accelerations during execution of the scaled position-lock cam.

Moving While Camming

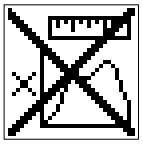
You can use an incremental Move Axis block on the slave axis (or master axis if configured for servo operation) while the position-lock cam is operating. This is particularly useful to accomplish phase advance/retard control. The incremental move distance can be used to eliminate any phase error between the master and the slave, or to create an exact phase relationship.

You can also use a phase shift move while the position-lock cam is operating to shift the master reference position of the cam on the fly. Unlike an incremental move on the slave axis, a phase shift move on the slave axis shifts the cam profile relative to the master axis, as shown below.



See *Move Axis* in this chapter for more information on phase shift moves.

Disable Position Lock CAM

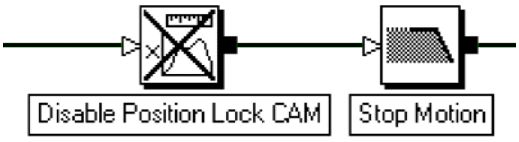


The Disable Position Lock CAM block stops the position-lock cam currently executing on the selected Axis. The axis is stopped immediately with no deceleration.

The Disable Position Lock Cam block resides on the CAM Palette.

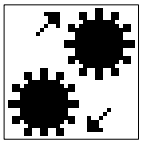
Stopping the PCAM Smoothly

When you select Continue Jog at last slave speed, the position-lock cam motion is stopped by blending it into a jog at the current speed of the axis. The axis can then be smoothly decelerated to a stop using a Stop Motion block, as shown below.



ATTENTION: If you select Continue Jog at last speed, the slave axis attempts to jog at the last actual slave velocity regardless of its own maximum velocity parameter.

Disable Gearing

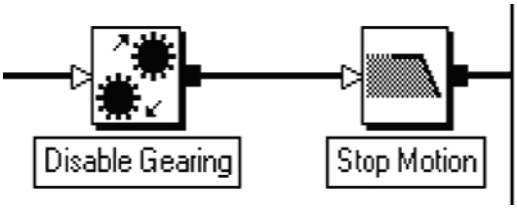


The Disable Gearing block immediately stops the electronic gearing motion of the selected Axis with no deceleration. If gearing is the only motion in progress on the axis, the axis stops.

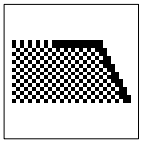
The Disable Gearing function block resides on the Main Palette.

Stopping Gearing Smoothly

When you select Continue Jog at last master speed, the electronic gearing motion is stopped by blending it into a jog at the current speed of the axis. The axis can then be smoothly decelerated to a stop using a Stop Motion block, as shown below.



Stop Motion



Use the Stop Motion block to stop all motion, or any specified type of motion, on the selected servo axis or interpolator. Except in Kill Control mode (see below), the Stop Motion block does not disable feedback.

The Stop Motion block resides on the Main Palette.

The Stop Motion block (except when Kill Control is selected) stops the selected type of motion by blending that motion into a jog at the current speed of the axis, then stopping the jog at the specified deceleration rate using the jog profile set in the Positioning page of the Configure Axis Use dialog box. You can use this block to stop a specific type of motion on an axis and leave other motion unaffected.

The Stop Motion block also (except when Kill Control is selected) immediately sets all appropriate motion status variables to 0 (false). However, Jog_status remains true (1) until the deceleration to zero speed is complete, at which time it too becomes 0 (false).

See *Axis Locked and Axis Done Conditions* chapter for a complete discussion of the operation of these status conditions when motion is stopped.

ATTENTION: Never enter a deceleration value of zero— or any expression or equation that equals zero— axis motion does not stop.

When stopping gearing or cams, select the slave axis to turn off the gearing or camming and stop the axis.

If the master axis is a servo axis (physical or imaginary), this block stops the master axis, which in turn stops the geared or cammed slave axis without disabling gearing or camming.

When stopping interpolated motion, all axes currently commanded by the interpolator stop at the accelerate/decelerate rate (set in the Interpolate Axes block that initiated the motion).

Kill Control

The Kill Control selection:

- stops all motion on all axes
- disables feedback on all axes

- resets all servo outputs to zero
- deactivates all drive enable outputs
- disables the CPU Watchdog
- aborts the program.

Use this block only in an emergency situation when the motion controller must be disabled quickly.

Note: The stopping rate of the axes depends on the type of amplifier and the specific connections utilized, because the motion controller no longer controls the axes after this block executes.

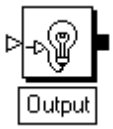
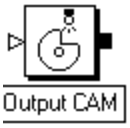
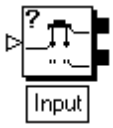
To re-enable the CPU Watchdog again, after executing a Stop Motion block, cycle power to the motion controller or press the front panel RESET button.

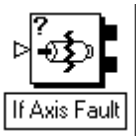
Stopping All Motion

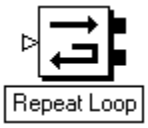
When you select Stop All, all motion caused by any previous Move Axis, Home Axis, Jog Axis, Gear Axes, Position Lock Cam, Time Lock Cam and/or Interpolate Axes block stops simultaneously, bringing the axis/axes to rest. If the specified axis is interpolating, the appropriate interpolator is stopped. This causes interpolated motion, on all axes commanded by the appropriate interpolator, to stop at the accelerate/decelerate rate specified in the Interpolate Axes block that initiated the motion.

I/O and Event Blocks

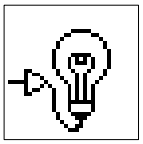
I/O and Event blocks control discrete I/O as well as registration, timer, and position events. The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Directly and immediately turn on or turn off a single Flex I/O, general purpose, RIO, SLC, or AxisLink discrete output.
	• Enable an output cam (also known as a programmable limit switch). • Disable an output cam enabled by a previous Output Cam block.
	• Pause the current task until a selected input is on (Wait for Input On) or off (Wait for Input Off). • Branch program flow to the 1 (true) node at the top of the block if input is on, or to the 0 (false) node at the bottom of the block if input is off (If Input).
	• Pause the current task until the status of a selected axis' or interpolator is true (Wait for Axis). • Monitor the status of an axis or interpolator. Program flow branches to the: •Top (true) node of the block if the selected axis or interpolator status matches the actual status •Bottom (false) node of the block if the selected status does not match the actual status (If Axis).

Use this block:	To:
	• Arm a watch position event to occur when the selected physical or imaginary axis reaches the setpoint position. • Disarm a previously armed watch position event that has not yet occurred. • Arm a registration event to store the actual positions of all physical and virtual axes, and the command position of the imaginary axis on the specified edge of a dedicated high-speed registration input.
	• Disarm a registration event, which has not yet occurred. • Enable a watch position event, to occur when the watch position event occurs. • Enable an event or action, to occur in response to a setpoint position event or to a dedicated or configured input event. • Disarm an event or action, which has not yet occurred.
	• Pause the current task until the watch position event, or the registration event, occurs on the selected axis. • Monitor a previously configured position or registration event, and branch program flow to the: •Top (true) node if the event has occurred •Bottom (false) node if the event has not occurred.
	• Set one of four count down timers to the desired time setting. • Set the value of the free running clock.
	• Pause the current task until the selected count down timer has timed out. • Monitor the selected count down timer to determine if it has timed out, and branch program flow to the: •Top (true) node if the timer has timed out •Bottom (false) node if the timer has not times out.
	Check for fault conditions on the axes.

Use this block:	To:
	Repeat the sequence—or loop—of blocks connected to this block's bottom output node and input node, a specified number of times.

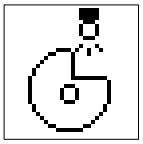
Output



The Output block directly and immediately turns on or off a single Flex I/O, general purpose, RIO, SLC, or AxisLink discrete output. Use the Tag Explorer field to select an output type. At the Tag Window select the specific output to turn ON or OFF.

The Output block resides on the Main Palette.

Output CAM



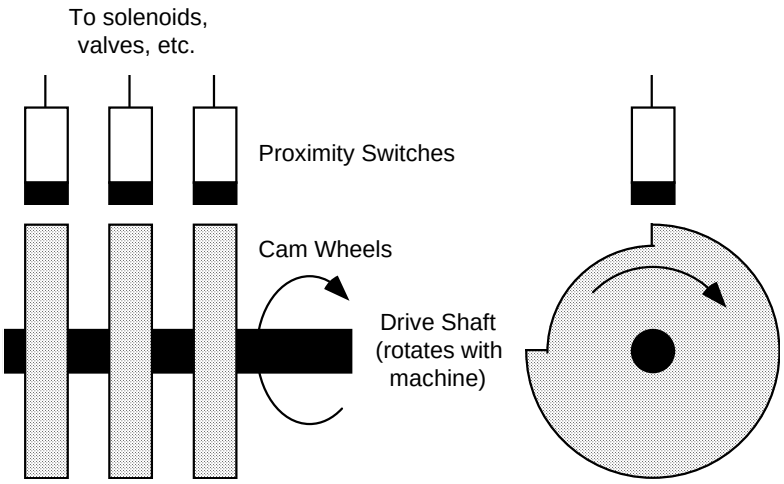
Use the Output Cam block to:
- Enable and Set up an output cam.
- Disable an output cam set up by a previous Output Cam block.

Output cams—also known as programmable limit switches—are discrete outputs that turn on at a specific axis position and off at another position. They are most often used to replace mechanical cams, and to synchronize the activation of solenoids and valves with the position of an axis. You can configure up to 48 separate output cams to control Flex I/O outputs, general purpose outputs, SLC output bits, or RIO output bits. By specifying the same output for more than one output cam, you can configure the output to cycle more than once per machine cycle.

The Output Cam dialog box resides on the CAM palette.

When you enable an output cam, the selected discrete output is no longer available for use in an Output block. When you disable an output cam, the output again becomes available for use in an Output block as a standard discrete output. Until the discrete output is re-used, and given a new value by the application program, it retains its current value.

A typical mechanical output cam consists of a shaft geared 1:1 with the machine such that it rotates exactly one revolution per machine cycle. Attached to this shaft are many individual cam wheels with cutouts that are read by proximity switches whose outputs are used to activate solenoids, valves, etc., as shown below.



The Output Cam block simulates this operation using a discrete output. Up to 48 output cams can be configured using Flex I/O outputs, general purpose outputs, RIO discrete outputs, or SLC output bits. Generally, when the position of the selected axis is within the window defined by the start of window and end of window positions, the output is on; otherwise, it is off.

ATTENTION: If a discrete output is configured as an output cam, do not use an Output block to turn that discrete output on or off.

The AxisLink virtual axes are functionally equivalent to physical master only axes, therefore you can select a virtual axis for output cams. However, only one of the two available virtual axes can be enabled at one time—enabling one virtual axis automatically disables the other virtual axis (if it was previously enabled). Before you use an AxisLink virtual axis for an output cam, you must first enable it with a Virtual Axis Control block.

A virtual axis has no command position because they are equivalent to physical master only axes. When you cam to a virtual axis, only Actual appears in the Position menu.

Camming to the Imaginary Axis

You can also use the imaginary axis as the axis for output cams. The output of an imaginary axis is its command position, it has no actual position, therefore only Command appears in the Position menu when camming to an imaginary axis.

Camming to Actual Position

When you select Actual from the Position menu, the actual position of the selected axis is used to generate the output cam transitions.

Actual position is the current position of a physical or virtual axis as measured by its encoder or other feedback device. This is the usual selection (the only available selection when the master axis is a master only axis, physical or virtual,) because it is necessary to activate devices based on the actual position of the axis.

Camming to Command Position

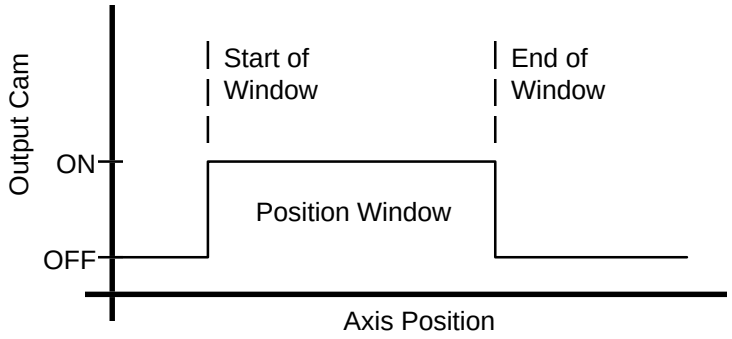
When you select Command from the Position menu, the command position of the selected axis is used to generate the output cam transitions.

Command position is only available when the selected axis is a servo or the imaginary axis and is the intended or commanded position of the axis, as generated by any previous motion blocks. Virtual axes being equivalent to physical master only axes have only Actual appear in the Position menu. (You cannot select Command from the Position menu, if you have chosen a virtual axis for the output cam.)

Command position is a more stable reference for camming, because it does not incorporate any associated following error or external position disturbances. When you use the command position, the selected axis must be commanded to move in order to cause any output cam transitions. See the *Installation and Setup* manual for your motion controller for more information on command position.

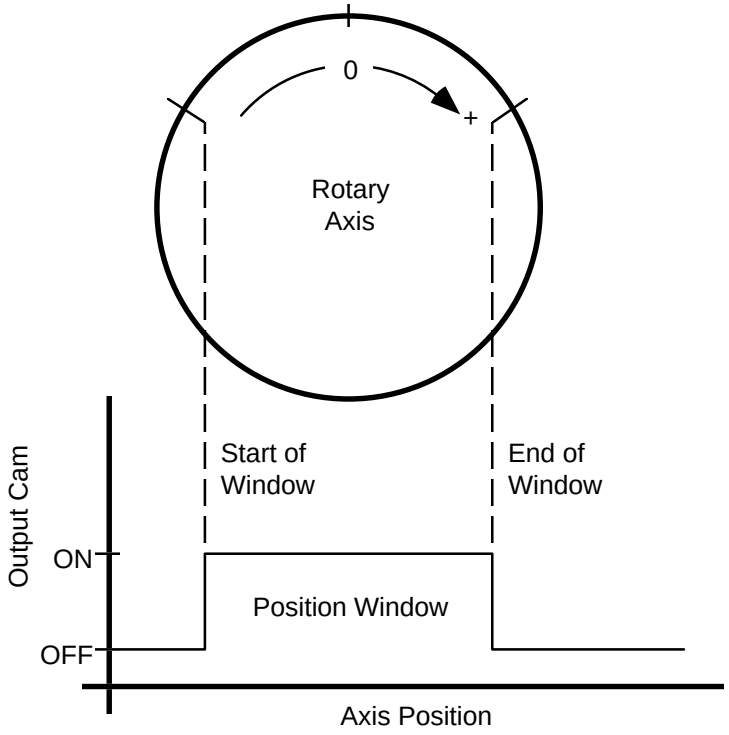
Defining the Window

The position window, within which the selected discrete output is on, is defined by the start of window and end of window values, as shown below.



For linear axes, both values may be positive (as shown above) or negative, or may be of different signs. However, the start of window value should be less than the end of window value, otherwise the output is never turned on.

For rotary axes, both values must be less than the value of the conversion constant divided by the unwind constant (as set in Configure Axis Use Feedback dialog box). The start of window value may be greater than the end of window value to allow output cams which cross the unwind point of the axis.

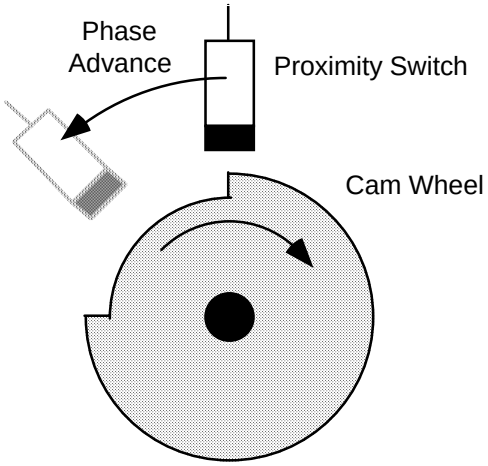


Actuation Delay

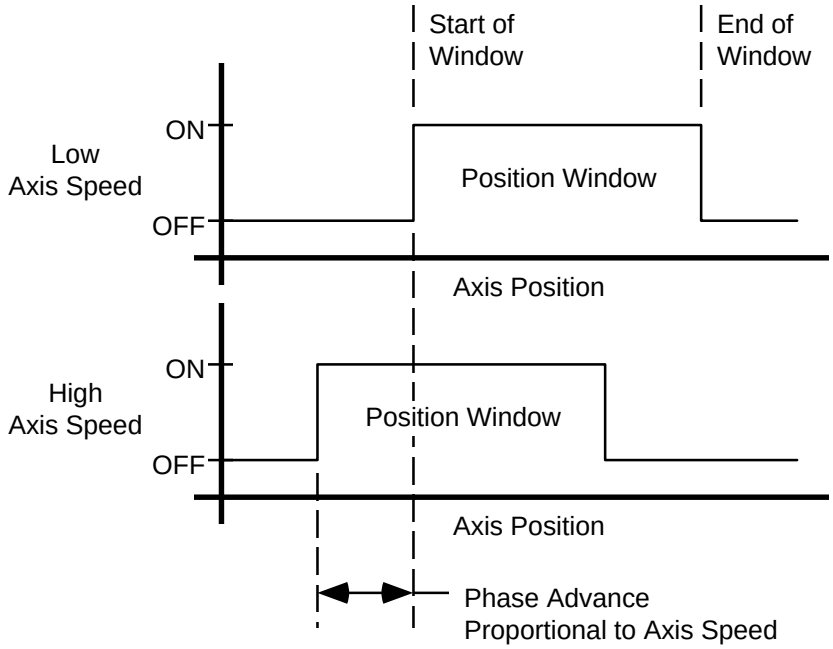
Solenoids, valves, relays, etc. have a delay between the following:

- the time when the electrical actuating signal is applied.
- the moment the device actually turns on.

You must consider this actuation delay time to ensure that the device actually turns on at the correct position, and is not merely commanded to turn on at the correct position. Mechanically, this compensation is accomplished by advancing (moving) the proximity switch array relative to the cams as a function of machine speed.



With output cams, the actuation delay value compensates for the intrinsic device actuation delay. Enter the actual actuation delay of the device in seconds. Make sure to include significant delays in any optical isolation circuitry. As the speed of the axis increases, the position window advances proportionally to the speed of the axis and the actuation delay value.



This ensures that the device turns on at the selected axis position regardless of axis speed.

Output cam with actuation delay is most accurate when the delay is short and the speed constant. Sharp acceleration near the point where the output must be actuated may cause inaccuracies in the timing of the output, and may even cause some flickering of the output as the axis enters and exits the cam window. Use command position rather than actual position whenever possible, because it is smoother.

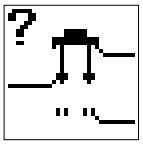
Tag Explorer

If you selected **Enable**, select the type of output.

Tag Window

If you selected **Enable**, select a discrete output.

Input



Use an Input block to execute one of these three commands:

- Wait for Input On pauses the current task until the selected Dedicated (axis-specific), Miscellaneous, or Configured (AxisLink, RIO Adapter, RIO Scanner, SLC, or Flex I/O discrete) input is on.
- Wait for Input Off pauses the current task until the selected Dedicated (axis-specific), Miscellaneous, or Configured (AxisLink, RIO Adapter, RIO Scanner, SLC, or Flex I/O discrete) input is off.
- If Input branches program flow to the top (true) node if input is on, or to the bottom (false) node if input is off.

The Input block resides on the Main Palette.

There are two pages to the Input block. The first page requires you to make entries in the **Type** and **Input Class** fields. The second page is dependent upon your selection for the Input Class field.

Type

The Type field determines the function to be performed. The three options are:

- Wait for Input On
- Wait for Input Off
- If Input

Require OFF to ON Transition

When you select Wait for Input ON, you enable the *Require OFF to ON transition* field.

If you select Require OFF to ON Transition, the program pauses until the selected input turns off and then on again. This is the same as using two Input blocks with the same parameter settings, except that:

- the first Input block would be set to Wait for Input OFF
- the second Input block would be set to Wait for Input ON
- the Require OFF to ON Transition would not be selected in either Input block.

Require ON to OFF Transition

When you select Wait for Input OFF, you enable the *Require ON to OFF Transition* field.

If you select Require ON to OFF Transition, the program pauses until the selected input turns on and then off again. This is the same as using two Input blocks with the same parameter settings, except that:

- the first Input block would be set to Wait for Input ON
- the second Input block would be set to Wait for Input OFF
- the Require ON to OFF Transition would not be selected in either Input block

Input Class

There are three options for the Input Class field:

- Configured – causes the Configured tab to display for access to the Configured page.
- Dedicated – causes the Dedicated tab to display for access to the Dedicated page.
- Miscellaneous – causes the Miscellaneous tab to display for access to the Miscellaneous page.

Configured

To complete the definition of a configured input, make entries in the following fields:

- **Tag Explorer** – Select an input type from the tree control. The specific inputs or input bits associated with a the selected input type appear in the Tag Window.

  Note: If an input type does not appear, it has not been enabled.

- **Tag Window** – Select the input or input bit to use with a Wait for Input On, Wait for Input Off, or If Input command.

  Note: If a particular input or input bit does not appear, it has not been defined.

Dedicated

To complete the definition of a dedicated input, make entries in the following fields:

- Axis – Select the axis, from which the input is read.
- Input – Select the desired Input

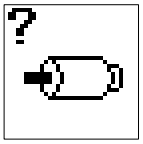
The available selections will depend upon:

- The Controller Type selected in the General page of the Configure Control Options dialog box (e.g., Resolver Loss Event and Resolver Loss State selections apply only to 1394 and 1394

Turbo controllers).

- The Axis selected, above (e.g., for AXIS1 on a 1394L controller, only the Registration, Encoder Loss, and Marker State selections apply).

On Axis



Use the On Axis block to execute the following commands:

- Wait for Axis pauses program flow until the status of a selected axis or interpolator is true.
- If Axis monitors the status of a selected axis or interpolator. Program flow branches to the top (true) node if the selected axis or interpolator status matches the actual status, and to the bottom (false) node if the selected status does not match the actual status.

The On Axis block resides in the Main Palette.

Wait for Axis and Multitasking

If more than one task is executing (multitasking), the Wait for Axis command halts the task containing this block, while the other tasks continue to execute. In this way, including this block in one task does not interrupt the execution of any other task, or hang the task dispatcher.

If Axis and Multitasking

If more than one task is executing (multitasking), you can use the If Axis command in one task to evaluate the status of an axis or interpolator whose motion was initiated by another task, even if Wait for Completion is selected in the block that initiated the motion.

Status

The available status selections vary, depending upon the type of command—Wait for Axis or If Axis, the selection of axis or interpolator, and the particular axis selected. The following table lists the available status selections depending upon these combinations. Although not listed below, note that the only available status selections for a master only axis are Homing (If Axis) or Homing Done (Wait for Axis).

Available Status Selections (S = Servo; V = Virtual; I = Imaginary)

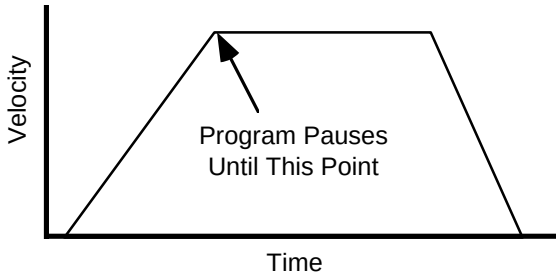
	Wait for Axis Mode				If Axis Mode			
	Axis				Axis			
Status	S	V	I	Interpolator	S	V	I	Interpolator
Accelerated	✔		✔	✔				
Accelerating	✔		✔	✔	✔		✔	✔
AxisLinked		✔				✔		
Decelerated	✔		✔	✔				
Decelerating	✔		✔	✔	✔		✔	✔
Feedback On	✔		✔		✔		✔	
Gearing Off	✔		✔					
Gearing On	✔		✔		✔		✔	
Gearing Opposite	✔		✔		✔		✔	
Homing					✔	✔		
Homing Done	✔	✔						
Interpolating								✔
Interpolation Done				✔				
Jogging					✔		✔	
Jogging Done	✔		✔					
Locked	✔				✔			

Available Status Selections (S = Servo; V = Virtual; I = Imaginary)								
	Wait for Axis Mode				If Axis Mode			
	Axis				Axis			
Status	S	V	I	Interpolator	S	V	I	Interpolator
Merge Pending								✔
Merging Done				✔				
Moving					✔		✔	
Moving Done	✔		✔					
Output Limited	✔				✔			

Accelerated

When you select Accelerated in Wait for Axis mode, the current task pauses until either of the following occurs as shown below:

- the selected servo axis is no longer accelerating (Accel_status = 0) due to a Move Axis or Jog Axis block
- the selected interpolator is no longer accelerating (Accel_status_Interp0 or …Interp1 = 0) due to an Interpolate Axes block

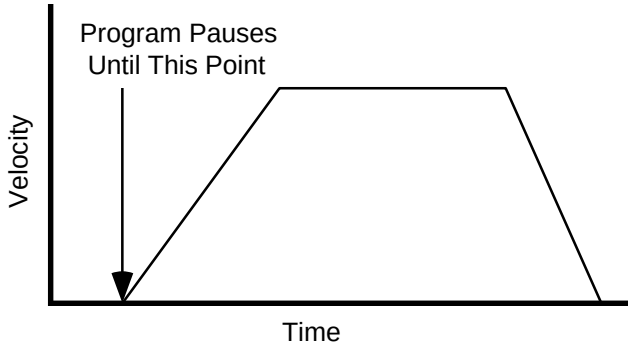


See *Move Axis*, *Jog Axis*, and *Interpolate Axes* for more information on commanding axis motion. Any acceleration caused by electronic gearing, time-lock cams, or position-lock cams does not affect the Wait for Axis accelerated status.

Accelerating

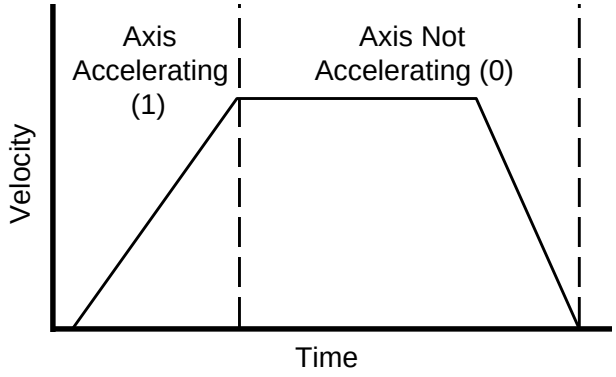
When you select Accelerating in Wait for Axis mode, the current task pauses until either of the following occurs as shown below:

- the selected servo axis is accelerating (Accel_status = 1) due to a Move Axis or Jog Axis block
- the selected interpolator is accelerating due to an Interpolate Axes block (Accel_status_Interp0 or …Interp1 = 1)



When you select Accelerating in If Axis mode, the current task branches to the top (true) node if either of the following occurs as shown below:

- the selected physical or imaginary axis is decelerating (Accel_status = 1) due to a Move Axis or Jog Axis block
- the selected interpolator is decelerating due to an Interpolate Axes block (Accel_status_Interp0 or …Interp1 = 1)



See *Move Axis*, *Jog Axis*, *Interpolate Axes* and *Stop Motion* for more information on moving, jogging, interpolating, and stopping axes. Any acceleration caused by electronic gearing, time-lock cams, or position-lock cams does not affect the accelerating status in the On Axis block.

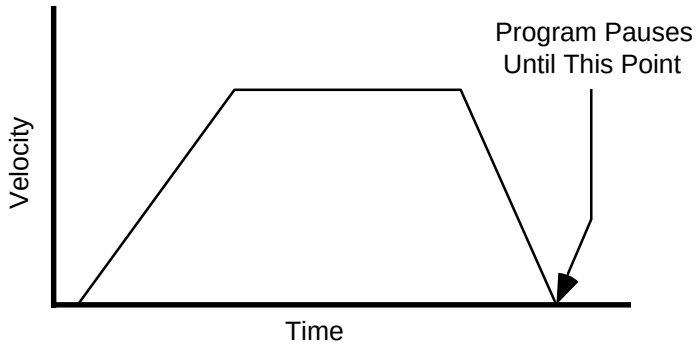
AxisLinked

When you select AxisLinked from the Status menu in:	The program:
Wait for Axis mode	Pauses until the selected virtual axis is enabled (AxisLink_status = 1).
If Axis mode	Branches to the top (true) node if the selected virtual axis is enabled (AxisLink_status = 1).

Decelerated

When you select Decelerated in Wait for Axis mode, the current task pauses until either of the following occurs:

- the selected servo axis is no longer decelerating (Decel_status = 0) due to a Move Axis or Jog Axis block
- the selected interpolator is no longer decelerating (Decel_status_Interp0 or …Interp1 = 0) due to an Interpolate Axes block

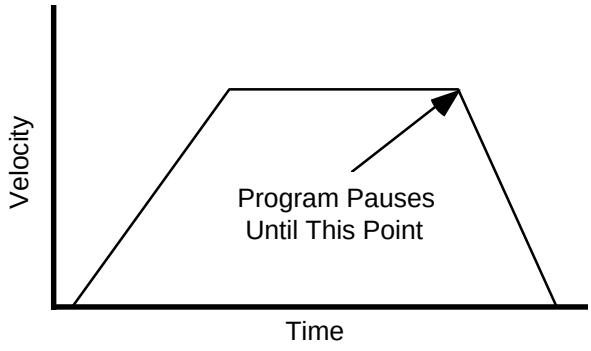


See *Move Axis*, *Jog Axis*, *Interpolate Axe*s, and *Stop Motion* for more information on moving, jogging, interpolating, and stopping axes. Any deceleration caused by electronic gearing, time-lock cams, or position-lock cams does not affect the decelerated status in the Wait for Axis block.

Decelerating

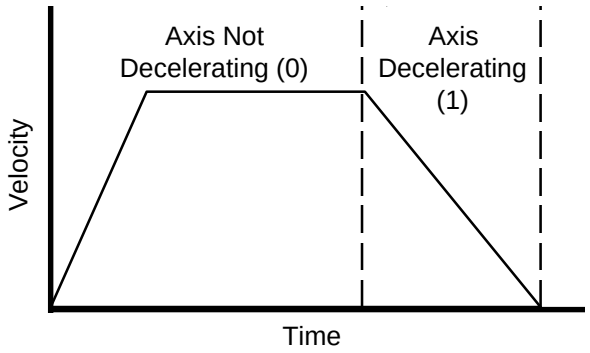
When you select Decelerating in Wait for Axis mode, the current task pauses until one of the following occurs:

- the selected servo axis is decelerating (Decel_status = 1) due to a Move Axis or Jog Axis block
- the selected interpolator is decelerating due to an Interpolate Axes block (Decel_status_Interp0 or …Interp1 = 1)



When you select Decelerating in If Axis mode, the program branches to the top (true) node if one of the following occurs as shown below:

- the selected servo axis is decelerating (Decel_status = 1) due to a Move Axis or Jog Axis block
- the selected interpolator is decelerating due to an Interpolate Axes block (Decel_status_Interp0 or …Interp1 = 1)



Feedback On

When you select Feedback On in:	The:
Wait for Axis mode	Current task pauses until feedback for the selected servo axis is on (Feedback_status = 1).
If Axis mode	Program branches to the top (true) node if feedback on the selected servo axis is on (Feedback_status = 1).

See the *Setup and Installation Manual* for your motion controller for a general discussion of the feedback loop used in the motion controllers.

Gearing Off

If you select Gearing Off in Wait for Axis mode, the current task pauses until electronic gearing on the selected servo axis is off (Gearing_status = 0).

Gearing On

When you select Gearing On in:	The:
Wait for Axis mode	Current task pauses until electronic gearing on the selected servo axis is on (Gearing_status = 1).
If Axis mode	Program branches to the top (true) node if electronic gearing on the selected servo axis is on (Gearing_status = 1).

Gearing Opposite

When you select Gearing Opposite in:	The:
Wait for Axis mode	Current task pauses until the direction of electronic gearing on the selected servo axis is opposite.
If Axis mode	Program branches to the top (true) node if the direction of electronic gearing on the selected servo axis is opposite.

Homing / Homing Done

When you select Homing in If Axis mode, the program branches to the top (true) node if the selected physical or virtual axis is currently being commanded to home (Homing_status = 1).

When you select Homing Done in Wait for Axis mode, the current task pauses until the selected servo axis is no longer being commanded to home (Homing_status = 0).

Interpolating / Interpolating Done

When you select Interpolator and status:	In:	The:
Interpolation Done	Wait for Axis mode	Current task pauses the selected interpolator is no longer causing axis motion on any axis (Interp0_status or Interp1_status = 0).
Interpolating	If Axis mode	Program branches to the top (true) node if the selected interpolator is currently commanding its axes to move (Interp0_status or Interp1_status = 0).

See *Interpolate Axes* section and *Stop Motion* section for information on interpolation and stopping interpolated motion respectively. See the *Axis Locked and Axis Done Conditions* chapter in this manual for more information about the interpolation done status condition.

Jogging/Jogging Done

When you select:	In:	The:
Jogging Done	Wait for Axis mode	Current task pauses until the selected physical or imaginary axis is done jogging (Jog_status = 0).
Jogging	If Axis mode	Program branches to the top (true) node if the selected physical or imaginary axis is currently being commanded to jog (Jog_status = 1).

See *Jog Axis* for information on jogging axes, and *Stop Motion*, for information on stopping jogs. See the *Axis Locked and Axis Done Conditions* chapter in this manual for further discussion of the jogging done status condition.

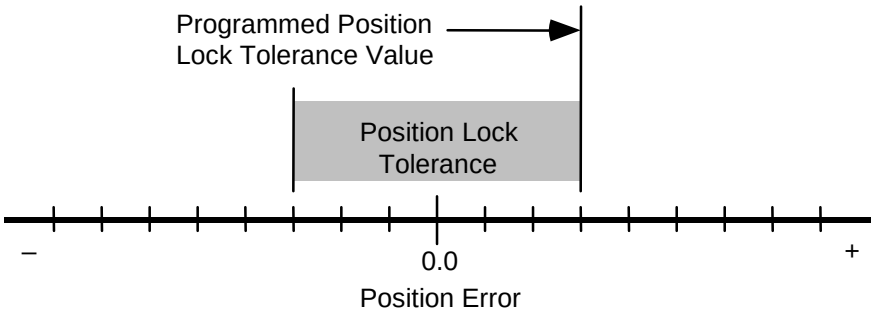
Locked

When you select Locked in:	The:
Wait for Axis mode	Current task pauses until the selected physical axis is locked onto its command position (Lock_status = 1).
If Axis mode	Program branches to the top (true) node if the selected physical axis is locked onto its command position (Lock_status = 1).

An axis is locked whenever either of the following two sets of conditions are true:

- Feedback on, electronic gearing off and no position-lock cam or interpolated motion in progress:
  - No motion (Home, Move, Jog, or Time-Lock Cam) in progress.
  - No axis faults (drive fault, position error, encoder noise or loss, hardware or software overtravel fault) exists.
  - The output voltage of the axis is not being limited to the servo output limit parameter (Servo Output Limited).
  - Position error is less than or equal to position lock tolerance.
- Feedback on, electronic gearing on or position-lock cam or interpolated motion in progress:
  - No motion (Home, Move, Jog, or Time-Lock Cam) in progress.
  - No axis faults (drive fault, position error, encoder noise or loss, hardware or software overtravel fault) exits.
  - The output voltage of the axis is not being limited to the servo output limit parameter (Servo Output Limited).
  - Position error is less that or equal to position lock tolerance.

The position lock tolerance is set in the Positioning page of the selected axis Configure Axis Use dialog box, and specifies how much position error the motion controller tolerates in a locked condition. It is one of the factors that determines positioning accuracy. The Position Lock Tolerance value is interpreted as a ± quantity.



See the *Setup* section of the *Installation and Setup Manual* for more information on position lock tolerance. See the *Axis Locked and Axis Done Conditions* chapter in this manual for further discussion of the axis locked status condition.

Merge Pending / Merging Done

When you select Merge Pending in If Axis mode, the program branches to the top (true) node if a new motion segment using the selected interpolator is currently waiting to be merged into the current axis motion (Merge_status_Interp0 or …Interp1 = 1).

When you select Merging Done in Wait for Axis mode, the current task pauses until the previous motion caused by the selected interpolator has been completely merged (Merge_status_Interp0 or …Interp1 = 0) into the new interpolated move.

Moving / Moving Done

When you select Moving in If Axis mode, the program branches to the top (true) node if the selected physical or imaginary axis is currently being commanded to move or is executing a time-lock cam (Move_status = 1).

When you select Moving Done in Wait for Axis mode, the current task pauses until the selected physical or imaginary axis is done moving (Move_status = 0).

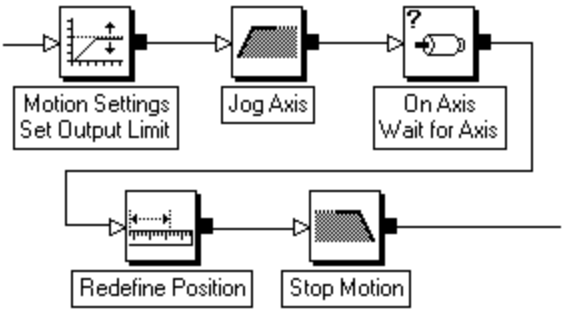
See *Move Axis* for information on moving axes and *Stop Motion* for information on stopping moves. See the *Axis Locked and Axis Done Conditions* chapter in this manual for further discussion of the moving done status condition. See *Move Axis*, *Time Lock Cam*, and *Stop Motion* for information on starting and stopping moves and cams.

Output Limited

When you select Gearing Opposite in:	The:
Wait for Axis mode	Current task pauses until the servo output of the selected physical axis is clamped at the pre-defined servo output limit as set in the Servo page of the selected axis Configure Axis Use dialog box, or via a previous Motion Settings block (Output_limit_status = 1).
If Axis mode	Program branches to the top (true) node if the servo output of the selected physical axis is currently clamped at the pre-defined servo output limit as set in the motion controller's machine setup menu or via a previous Motion Setting block (Output_limit_status = 1).

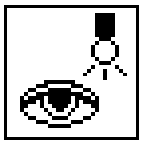
You can use the output limited condition, in conjunction with the Motion Settings block, to detect intended stall conditions on an axis, such as moving or homing an axis to a hard stop.

For example, this GML sequence first sets the servo output limit to a low value to limit the force applied to the stop, then jogs the axis at a low speed and with low acceleration rates until a positive stop is detected (servo output limited).



Once the stop is engaged, the axis position is then re-defined to be the known position of the stop and the motion is stopped.

Watch Control

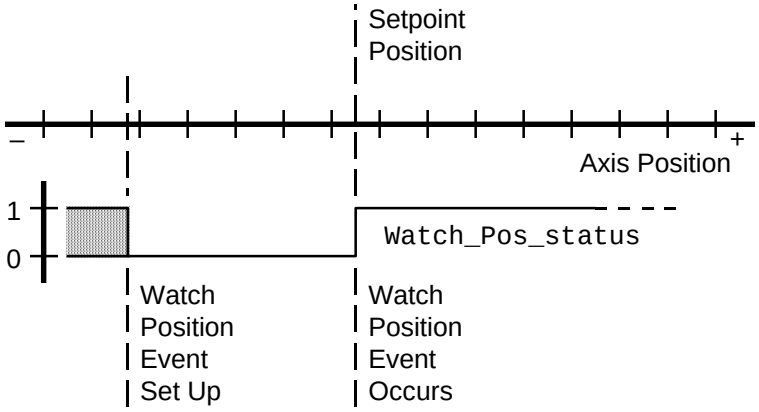


Use the Watch Control block to:

- Arm a watch position event to occur when the selected physical or imaginary axis reaches the setpoint position. (See *Arm Watch Position*.)
- Disarm a previously armed watch position event that has not yet occurred. (See *Disarm Watch Position*.)
- Arm a registration event to store the actual positions of all physical and virtual axes, and the command position of the imaginary axis on the specified edge of a dedicated high-speed registration input. (See *Arm Registration*.)
- Disarm a registration event, which has not yet occurred. (See *Disarm Registration*.)
- Enable a watch position event to occur in response to a setpoint position event. (See *Enable Watch Position Event/Action*.)
- Enable a dedicated or configured input event to occur in response to a setpoint position event. (See Enable Input/Event Action.)
- Disarm an event or action, enabled by a previous watch control block, which has not yet occurred. (See Disarm Input/Event Action.)

The watch control block resides on the Main Palette.

The Watch Control block, with Arm type and Watch Position class selected, sets up a watch position event that occurs when the selected physical or imaginary axis reaches the specified position.



You can use watch position events to synchronize an operation to a specific axis position while the axis is moving, such as activating a solenoid to push a carton off a conveyor at a certain axis position.

When a Watch Control block with Arm and Watch Position selected executes the following:

- the Watch_Pos_status variable for the axis is set to 0 (false)
- the actual position of a physical axis or the command position of the imaginary axis is monitored (at the servo loop update rate) until it reaches the specified setpoint (watch) position

After the watch position event occurs, the Watch_Pos_status variable for the axis is 1 (true).

You can include multiple watch position events in a single program—but only one for each physical or imaginary axis on the motion controller. Each event is monitored independently and may be checked using the appropriate Watch_Pos_status variable.

Wait for Tripped

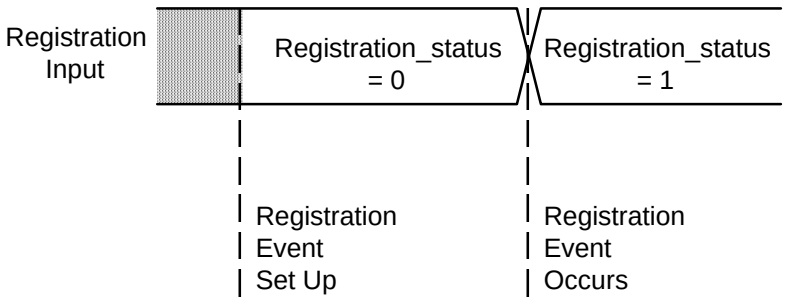
If you select Wait for Tripped, the program pauses until the watch position event occurs. When it does, the program continues with the next block. If other tasks are executing (multitasking), the task that contains this block pauses but the other tasks continue to execute. In this way, a Wait for Tripped selection in one task does not pause execution of any other tasks or hang the task dispatcher.

When Wait For Tripped is not selected, the Watch Control block, with Arm and Watch Position selected, sets up the watch position event and the program_continues with the subsequent blocks while the watch position event is monitored in the background.

The Watch Control block, with Arm type and Registration class selected, sets up a registration event to store the actual positions of all physical and virtual axes, and the command position of the imaginary axis on the specified edge of the dedicated high speed registration input for the selected physical or virtual axis.

When a Watch Control block with Arm and Registration selected executes, the registration input for the selected axis is monitored until a transition of the selected type (the registration event) occurs. When it does, the actual position of the axis is stored in the appropriate Registration_Position variable. In addition to this hard registration, the actual positions of all physical and virtual axes as well as the command position of the imaginary axis are also stored in the appropriate Soft_Reg_Pos*xx* variables when the registration event occurs. This soft registration stores the positions of all axes as sampled at the beginning of the first servo update following the registration event on the selected axis. See the *System Variables* chapter in this manual for more information on the Registration_Position and Soft_Reg_Pos*xx* variables.

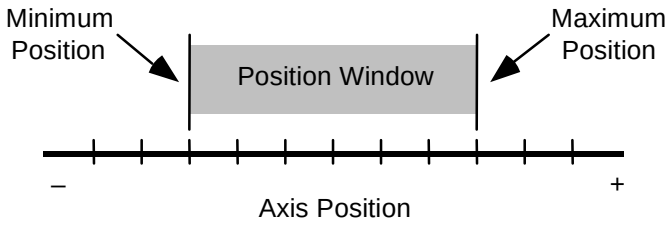
After the Watch Control block, with Arm and Registration selected executes, but before the selected transition occurs, the Registration_status variable for the axis is 0 (false), as shown below.



After the registration event occurs, the Registration_status variable for the axis is 1 (true). See the *System Variables* chapter of this manual for more information on variables.

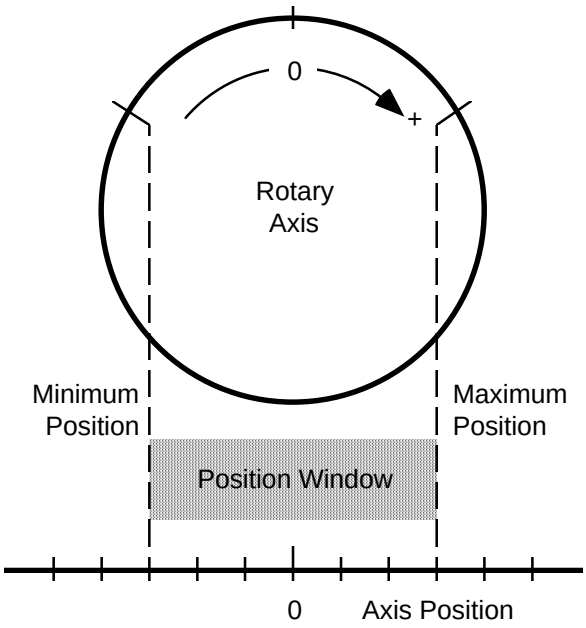
Windowed Registration

When you select Use Window on the Registration page, the selected trip state causes a registration event only if it occurs when the axis is within the window defined by the minimum and maximum positions, as shown below.



Enter values or expressions for the desired absolute positions that define the position window, within which the selected trip state of the registration input is valid. Windowed registration is useful to ignore spurious or random transitions of the registration sensor and improve the noise immunity of high-speed registration inputs.

For linear axes, the values can be positive, negative, or a combination. However, the minimum position value must be less than the maximum position value for the registration event to occur. For rotary axes, both values must be less than the resultant value derived from dividing the conversion constant by the unwind constant (both set in Feedback page of the Configure Axis Use dialog box). The minimum position value can be greater than the maximum position value for registration windows that cross the unwind point of the axis.



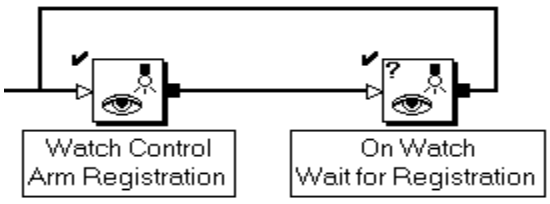
Wait for Tripped

When you select Wait for Tripped, the program pauses until the registration event occurs. When it does, the program continues with the next block. If other tasks are executing (multitasking), the task which contains this block pauses but the other tasks continue to execute. In this way, a Wait for Tripped selection in one task does not pause execution of any other tasks or hang the task dispatcher.

If you did not select Wait for Tripped, the Watch Control block with Arm and Registration selected sets up the registration event, and the program continues with the subsequent blocks while the registration event is monitored in the background.

Auto Rearm Input

When you select Auto Rearm Input, the registration event is automatically re-configured as specified in the Watch Control block with Arm and Registration selected, whenever a registration event occurs on the selected axis. This auto-registration is equivalent to—but much faster than—the following GML blocks executing in one task, while the latched registration position is used in another task. With Auto Rearm Input selected, Registration_status changes from 0 to 1 when the first registration event occurs and it stays at that value thereafter.



ATTENTION: You must enable Auto Rearm Input for any axis that is selected as a registering axis in an auto correction position-lock cam configuration.

Registration on Virtual Axes

AxisLink virtual axes are functionally equivalent to physical master only axes therefore, you can also use them for registration events. However, the registration input of the associated physical axis (on another motion controller or ALEC) is actually used to generate the registration event.

Only one of the two available virtual axes can be enabled at one time—enabling one virtual axis automatically disables the other virtual axis if it was previously enabled. Thus, to set up a registration event on an AxisLink virtual axis, the virtual axis must first be enabled using a previous Virtual Axis Control block.

Disarm Registration

The Watch Control block with disarm type and registration class selected cancels a registration event set up by a previous Watch Control block with Arm Registration selected that has yet to occur. If you selected Auto Rearm Input in the original Watch Control block (with Arm type and Registration class also selected), auto-registration also is disabled.

Enable Watch Position Event/Action

The Watch Control block, with Enable Event/Action type and Watch Position class selected, sets up a watch position event and a specific action that happens when the watch position event occurs. You can use Watch Position events to synchronize an operation to a specific axis position while the axis is moving, such as activating a solenoid to push a carton off a conveyor at a certain axis position.

To specify the action that happens when the watch position event occurs, one of the following blocks must be directly connected to the top (true) node of the Watch Control block:

- Gear Axes
- Position Lock Cam
- Redefine Position
- Move Axis

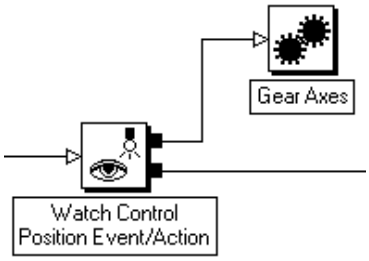
Blocks connected to the top (true) node of the Watch Control block have no output node since only a single action can happen when the watch position event occurs.

If the specified condition is not true when the Watch Control block is executed, program flow branches to the bottom (false) node of the block. When the selected watch position event occurs (regardless of what part of the diagram is executing), the block connected to the top (true) node of the Watch Control block is immediately executed.

You can activate multiple watch position events—one for each physical, virtual, or imaginary axis—at any given time. However, only one event action (either an input event action or a position event action) is allowed per axis. Either a Watch Control block, with Enable Event/Action type and Watch Position class selected, or a Watch Control block, with Arm type and Watch Position class selected, can be used on a given axis, but not both blocks. Configuring a second position type block on the same axis supersedes the first block. Each position event is monitored independently and can be checked using an On Watch block with Wait for Position Event selected, a Show Axis Position block, or an On Watch block with If Position Event selected. Watch position events do not automatically reactivate when tripped. For repetitive conditions, reactivate the watch position event with a new Watch Control block.

A Watch Control block, with Enable Event/Action type and Watch Position class selected, is similar to a Watch Control block, with Arm type and Watch Position class selected, followed by an On Watch block, with Wait for Position Event selected, except that the event action occurs much faster. In fact, all watch position event actions are designed to take effect within two (2) servo update periods of the occurrence of the watch position event. Furthermore, the action occurs automatically while execution of the diagram continues, without the need for additional blocks to monitor or wait for the event.

For example, the GML diagram below gears two axes when the axis passes a certain position.



Event Action Computations

Watch position actions are fast because the event action block (connected to the top (true) node of the Watch Control block) is completely evaluated when the Watch Control block executes, and is ready for instant response when the watch position event occurs. Therefore, any variables used in the event action block are evaluated once, immediately when the Watch Control block is executed. The event action block should not use any expressions that depend on the occurrence of the watch position event.

Enable Dedicated and Configured Input Event/Action

The Watch Control block, with Enable Event/Action type and Dedicated or Configured class selected, sets up an input event and a specific action that happens when the input event occurs. This allows the input event to directly cause an action when triggered.

To specify the action that happens when the input event occurs, one of the blocks shown below must be directly connected to the top (true) node of the Watch Control block:

- Gear Axes
- Position Lock Cam
- Redefine Position
- Move Axis
- Equation
- Interrupt SLC

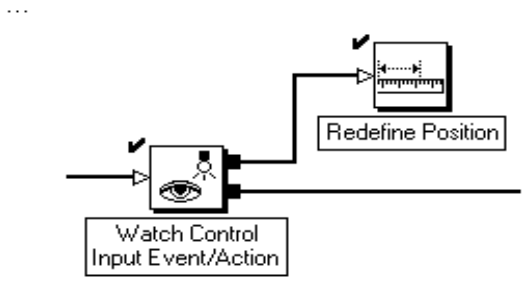
If you select the Equation block as the action block, you can only use it to save the actual or command position of an axis to a user variable.

If the selected condition is not true when the Watch Control block executes, program flow branches to the bottom (false) node of the block. When the selected input event occurs (regardless of what part of the diagram is executing), the block connected to the top (true) node of the Watch Control block immediately executes.

Multiple input events (one for each physical axis) can be active at a given time since each event is monitored independently. Additionally, one Flex I/O input and one SLC I/O input (1394T) can be active at a given time, thereby making it a total of five input event actions active at a time. However, only one event action (either an input event action or a position event action) is allowed per axis.

A Watch Control block with Enable Event/Action type, and Dedicated or Configure class selected, is similar to an Input block with Wait for Input ON or Wait for Input OFF, except that the event action occurs much faster. In fact, all input event actions are designed to take effect within two (2) servo update periods from when the input is detected. Furthermore, the action occurs automatically while execution of the diagram continues, without the need for additional blocks to monitor or wait for the event.

For example, the following GML diagram redefines the position of an axis when an input event is detected.



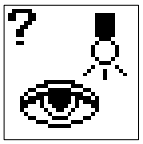
Input event actions are fast because the event action block (connected to the top (true) node of the Watch Control block) is completely evaluated when the Watch Control block executes and, is ready for instant response when the input event occurs. Therefore, any variables used in the event action block are evaluated once, immediately when the Watch Control block executes. The event action block should not itself use any expressions that depend on the occurrence of the input event.

Disarm Input Event/Action

The Watch Control block with Disarm type and Input Event/Action class selected cancels an input event set up by a previous Watch Control block with Enable Event/Action selected, that has not yet executed.

If this is input type is selected:	An:
Dedicated	Axis menu lets you select the desired physical axis.
SLC Signal	Event menu lets you select the desired event.

On Watch



Use the On Watch block to:

- Pause the program until the watch position event, or the registration event, occurs on the selected axis.
- Monitor a previously configured position or registration event, and branch program flow to the top (true) node if the event has occurred or to the bottom (false) node if the event has not occurred.

The On Watch block resides on the Main Palette.

Wait for Position Event

The Wait for Position Event type pauses the program until the watch position event for the selected axis occurs. The On Watch block with Wait for Position Event selected executes only if a watch position event has previously been set up for the selected axis, by a Watch Control block with Arm type and Watch Position class selected.

If you selected Wait for Tripped in the Watch Control block, do not use an On Watch block, with Wait for Position Event selected. If you do, the program waits for the same event twice.

The Wait for Position Event type requires no parameters, simply select the desired physical or imaginary axis.

If other tasks are executing (multitasking), the task which contains this block pauses, but the other tasks continue to execute. In this way, a Wait for Position Event selection in one task does not pause execution of any other tasks or hang the task dispatcher.

If Position Event

The If Position Event type evaluates a previously configured watch position event for the selected axis, without pausing the program. The If Position Event type executes only if a watch position event has previously been set up for the selected axis, using Watch Control block with Arm type and Watch Position class selected.

When the program executes, program flow branches to the top (true) node if the event has occurred, or to the bottom (false) node if the event has not occurred.

When multiple tasks are executing (multitasking), you can use an If Position Event-type in one task to evaluate a watch position event set up in another task, even if Wait for Tripped is selected in the Watch Control block. See *Watch Control* for more information on watch position events.

Wait for Registration

The Wait for Registration type pauses the program until the registration event for the selected axis occurs.

The Wait for Registration Event type executes only if a registration event has previously been set up for the selected axis, using a Watch Control block with Arm type and Registration class selected.

If you selected Wait for Tripped in the Watch Control block, do not use an On Watch block, with Wait for Registration selected. If you do, the program waits for the same event twice.

If other tasks are executing (multitasking), the task which contains this block pauses but the other tasks continue to execute. In this way, a Wait for Registration selection in one task does not pause execution of any other tasks or hang the task dispatcher.

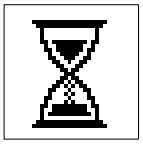
If Registration

The If Registration type evaluates a previously configured registration event for the selected axis, without pausing the program. An On Watch block with If Registration selected executes only if a registration event has previously been set up for the selected axis with an Arm Registration block.

When the program executes, program flow branches to the top (true) node if the event has occurred, and to the bottom (false) node if the event has not occurred.

When multiple tasks are executing (multitasking), you can use an If Registration type in one task to evaluate a registration event set up in another task, even if Wait for Tripped is selected in the Watch Control block. See *Watch Control* for more information on registration events.

Set Timer



Use the Set Timer block to:

- Set one of four count down timers to the desired time setting.
- Set the value of the free running clock.

The Set Timer block resides on the Main Palette.

Set Count Down Timer

The count down timer counts downward from the set time to zero. Use the count down timer to implement dwells and other time-related events in the diagram.

Wait for Timeout

Selecting one of the countdown timers displays the *Wait For Timeout* field.

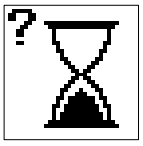
If you select Wait for Timeout, program execution halts until the timer times out. When it does, the program continues with the next block. If other tasks are executing (multitasking), the task which contains this block pauses but the other tasks continue to execute. In this way, a Wait for Timeout selection in one task does not pause execution of any other tasks or hang the task dispatcher.

If you do not select Wait for Timeout, the Set Timer block just sets up the timer and the program continues with the subsequent blocks while the timer counts down (or up) in the background.

Set Free Running Clock

The free-running clock runs up from set time. Use the free-running clock for measuring the execution time of program events, and to reset the set time (typically to zero) at the end of one event (or loop).

On Timeout



Use the On Timeout block to:

- Pause the program until the selected count down timer has timed out.
- Evaluate the selected count down timer to determine if it has timed out.

The On Timeout block resides on the Main Palette.

If Timeout

The If Timeout type evaluates the selected count down timer to determine if it has timed out. Program flow branches to the top (true) node if the timer has timed out, and to the bottom (false) node if the timer is still counting.

The selected timer must have been set up previously using a Set Timer block. See *Set Timer* in this chapter for more information on setting up timers.

When multiple tasks are executing (multitasking), you can use an If Timeout block in one task to evaluate a timer set in another task, even if you selected Wait for Timeout in the Set Timer block.

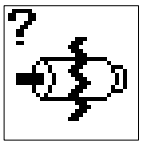
Wait for Timeout

The Wait for Timeout selection pauses the program until the selected count down timer has timed out.

The selected timer must have been set up previously using a Set Timer block. If you selected Wait for Timeout in the Set Timer block, do not use an On Timeout block with Wait for Timeout selected, otherwise the program waits for the timeout twice.

If other tasks are executing (multitasking), the task which contains this block pauses but the other tasks continue to execute. In this way, a Wait for Timeout selection in one task does not pause execution of any other tasks or hang the task dispatcher.

If Axis Fault



Use the If Axis Fault block to check for fault conditions on the axes. Using this block you can check:

- all axes for any faults all at once.
- a specific axis for any faults.
- a specific axis for a specific fault.
- AxisLink for a general Fault

The If Axis Fault block resides on the Main Palette.

Program flow branches to the top (true) node if a fault is active, and to the bottom (false) node if no fault is active.

The table below displays the axis faults which cause program flow to branch to the top (true) node of the If Axis Fault block and the corresponding message which appears in an axis status field in the runtime display (if enabled). See *Configure Auto Display* for information on defining the runtime display. See the *Fault Variable* descriptions for more information on each of the faults.

Axis Faults

Description	Runtime Display	S	MO	V	I
AxisLink Timed Out	AXL FLT			✔	
AxisLink Failed (axis not found)	AXL FLT			✔	
Drive Fault	DRV FLT	✔			
Position Limit Exceeded	ERR FLT	✔			
Hardware Overtravel Fault	HRD LIM	✔			
Software Overtravel Fault	SFT LIM	✔			
Encoder Loss or Noise Fault	ENC FLT	✔	✔		

Note: In the table above, the columns at the right show whether each fault value is valid for servo (S), master only (MO), virtual (V), or imaginary (I) axes.

The axis faults are prioritized from highest to lowest in the order shown in the previous table. A hardware overtravel fault has higher priority than a software overtravel fault. When a given fault is active, another fault of lower priority can also be active. For instance, if a hardware overtravel fault is active for an axis, a software overtravel condition may also be active on the axis. When the highest priority fault is cleared, the next highest priority fault is evaluated by a subsequent If Axis Fault block.

Checking for Faults on Any Axis

When you select Any Fault Any Axis, program flow branches to the top (true) node if any of the faults shown in the previous table are active on any axis. This selection is useful for building a global axis fault handling routine.

Checking for Any Fault on a Specific Axis

When you select Any Fault Specific Axis, program flow branches to the top (true) node if any of the faults shown in the previous table are active on the selected axis.

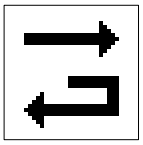
Checking for Specific Faults on an Axis

When you select Specific Fault, program flow branches to the top (true) node only if the fault, selected from the Fault drop-down menu, is active on the selected axis.

Checking for an AxisLink General fault

When you select AxisLink General Fault, program flow branches to the top (true) node if an AxisLink general fault has occurred.

Repeat Loop

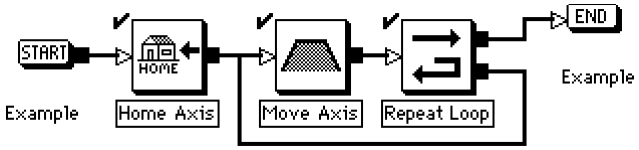


The Repeat Loop block repeats the sequence of blocks looping from its bottom output node to its input node the specified number of times. After executing this sequence of blocks the specified number of times, program flow branches to the output node at the top of the block.

The Repeat Loop block resides on the Main Palette.

Up to 1 billion (1,000,000,000) repeats may be specified.

The example diagram below homes an axis and then moves it repeatedly by an incremental distance a specified number of times.



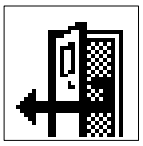
Program Control Blocks

Program Control blocks provide a convenient way of intelligently restarting and ending the application program. Blocks for restarting and ending the program are included in this section.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	• Restart the application program, causing the diagram to begin again from the Start block (Restart Type). • Provide a marker in the diagram that leads to other program blocks, which must be executed prior to ending the GML Commander program, when a fault occurs or when the program is intentionally stopped (When Restart Type).
	• Stop the execution of your GML Commander program (End Type). • Provide a marker in the diagram that leads to the execution of any required blocks before ending the program, when a fault occurs or when the program is intentionally stopped (When End or Fault Type).

Restart Program



Use the Restart Program block to:

- Restart the application program from the Start block, thereby allowing the diagram to re-boot itself (Restart Type).
- Provide a marker in the diagram to allow the program to be restarted from a location other than the Start block (When Restart Type).

The Restart Program block resides on the Main Palette.

Restart Type

When the program is restarted using the Restart selection, multitasking is turned OFF, all loop counters are set to zero, and the subroutine call stack is reset. This ensures that the diagram restarts with all critical program control registers in a known state. The program then restarts from a Restart Program block, with When Restart type selected (if the diagram contains one) or from the 

block (if there is no Restart Program block with When Restart type selected).

This block, with Restart type selected, has no output node. There is no direct output from this block because program flow branches directly either to the start block, or to a Restart Program block with When Restart selected, at execution of this block. In many cases, the lack of an output node on this block means that there is no connection to the 

block in the diagram. This missing connection generates a warning when the diagram is translated. In most cases this warning can be ignored .

Although the GML Commander diagram may contain any number of Restart Program blocks with Restart selected (in order to restart the program from many different places), the diagram can contain only one Restart Program block with When Restart selected.

When Restart Type

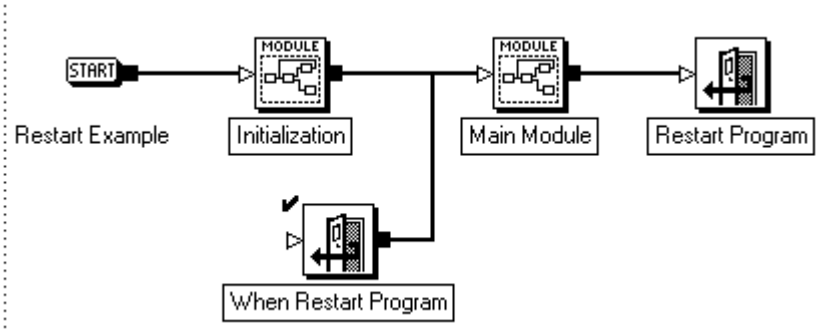
The Restart Program block, with When Restart selected, provides a marker in the diagram to allow the program to be restarted from a location other than the start block. If you include this type of block in your GML Commander diagram, program flow branches to it, rather than to the 

block, upon execution of a Restart Program block with Restart selected. This is useful, if certain conditions must be set up (resetting parameter or variable values, setting discrete outputs, stopping motion in progress, etc.) prior to restarting the program.

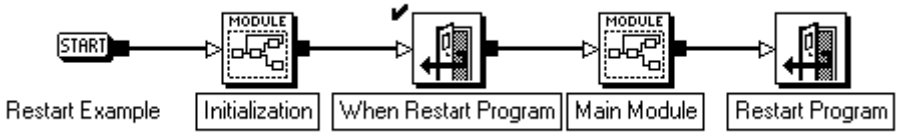
ATTENTION: Do not use more than one Restart Program block with When Restart selected in a GML Commander diagram.

The Restart Program block, with When Restart selected, must be used with another Restart Program block with Restart selected. Otherwise a Restart Program block with When Restart selected does nothing. A GML Commander diagram can contain any number of Restart Program blocks with Restart selected, to restart the program from many different places, but the diagram can contain only one Restart Program block with When Restart selected.

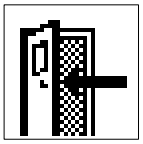
The following diagram shows how a Restart Program block with When Restart selected can be used to bypass an initialization module when restarting the program.



The Restart Program block with When Restart Program selected has an input node. It can be used in-line in a diagram, as it does nothing unless it is called by a Restart Program block with Restart selected. Thus, the following diagram is equivalent to the example above, but the function of the Restart Program block with When Restart selected is less obvious.



End Program



Use the End Program block to:

- Stop the execution of your GML Commander program.
- Provide a marker in the diagram that leads to other program blocks, which must be executed prior to ending the GML Commander program, when a fault occurs or when the program is intentionally stopped.

The End Program block resides on the Main Palette.

End Type

The End Program block, with End selected, stops the application program.

When you stop execution of the program using an End Program block with End type selected, program flow immediately branches either to another End Program block with When End or Fault type selected (if the diagram contains a When End or Fault type block) or to the 

block if there is no When End or Fault type block in the diagram.

The End Program block, with End type selected, has no output node. None is needed, because program flow branches directly either to the 

block, or to an End Program block, with When End or Fault selected, when the block executes. In some cases, the lack of an output node on this block means that there is no connection to the 

block in the diagram. This missing connection generates a warning when the diagram is translated which, in most cases, can be ignored.

A diagram, in order to stop the program from many different places, can contain any number of End Program blocks with End type selected. There can be only one End Program block with When End or Fault type selected in the diagram.

An End type block, in the When End or Fault sequence, unconditionally stops all tasks and ends the program.

When End or Fault Type

Select the When End or Fault option to provide a marker in the diagram that leads to the execution of any required blocks when a fault occurs, or when the program is intentionally stopped. A diagram can contain only one End Program block, with When End or Fault selected.

An End Program block with When End or Fault selected enables two options:

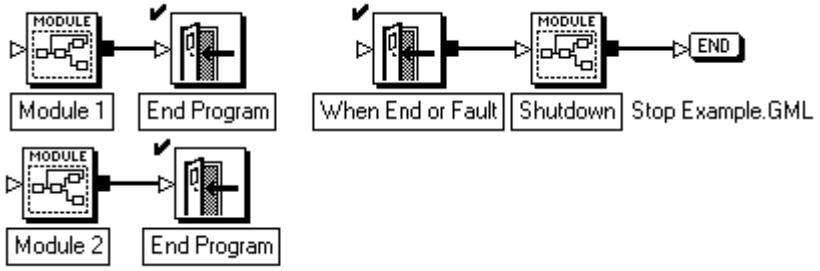
- Go to When End if Runtime Fault Occurs.
- Go to When End if Global Fault Occurs.

The Go to When End if Runtime Fault Occurs option reroutes program flow to this block when a runtime fault occurs during program execution.

The Go to When End if Global Fault Occurs option reroutes program flow to this block when a global fault occurs during program execution.

Program flow does not branch to the End Program block, with When End or Fault selected, when a fault occurs. An End Program block with When End or Fault selected, but with neither Go to When End… option selected, does nothing unless the program also contains at least one End Program block with End selected.

For example, the following diagram shows how an End Program block with When End or Fault selected can be used to execute a common shutdown routine when stopping the program from either of two places.



If you selected, in the End Program block with When End or Fault selected, either of the following the shutdown routine also executes if a runtime or global fault (respectively) occurs.

Runtime Faults

A diagram that includes an End Program block with When End or Fault and Go to When End if Runtime Fault Occurs selected, causes the program flow to branch to the End Program block with When End or Fault selected, rather than flow directly to the 

block, when either of the following occurs:

- a runtime fault occurs (Runtime_fault ≥ 0)
- an End type block executes

This is useful when runtime faults must be intercepted and handled within the application program (generated from the GML Commander diagram and downloaded to the motion controller) to avoid having to reset the motion controller to restart the application. If multitasking is ongoing when a runtime fault occurs, all tasks (including the main task) are suspended automatically as if a Task Control block with Stop Dispatcher selected had been executed.

See *Handling Faults in GML Commander,* for more information on recovering from a runtime fault and restarting the task dispatcher.

Global Faults

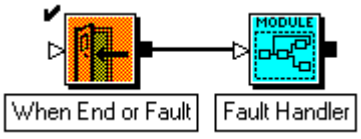
A diagram that includes an End Program block with both When End or Fault and Go to When End if Global Fault Occurs selected, causes the program flow to branch to the When End or Fault type block when either of the following occurs:

- a global fault occurs (Global_fault ≥ 0)
- an End type block executes.

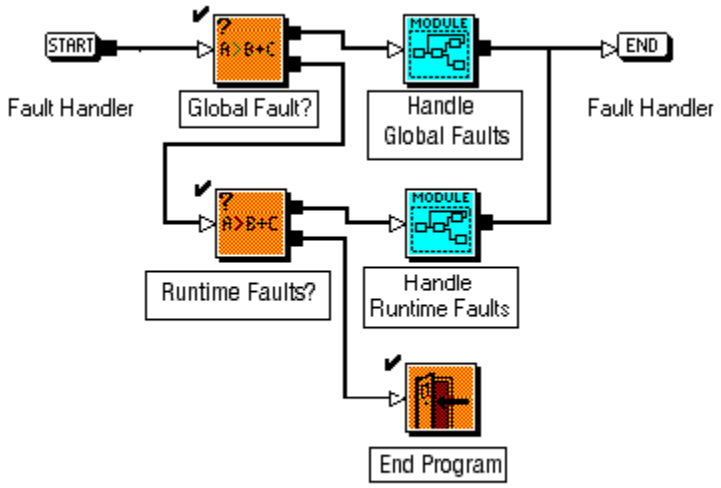
This provides a convenient mechanism for building a fault handling routine. If multitasking is ongoing when a global fault occurs, all tasks (including the main task) are suspended automatically as if a task control block with stop dispatcher selected had been executed.

Handling Faults in GML Commander

Using a When End or Fault type block with both Go to When End if Runtime Fault Occurs and Go to When End if Global Fault Occurs selected lets you handle all possible faults in the motion controller in a single fault handler module, as shown below:



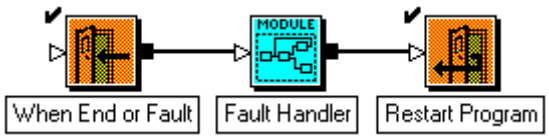
The fault handler module must determine if the End Program block, with When End or Fault selected, was executed due to a global fault, a runtime fault, or an End type block elsewhere in the diagram, and then take the appropriate action, as shown below.



ATTENTION: An End Program block, with End selected, in the fault handler unconditionally ends the program.

Unlike an End Program block with End selected elsewhere in the diagram (which executes the When End or Fault type block), an End Program block with End selected in the fault handler itself unconditionally stops all tasks and ends the program. Thus, it is imperative that you use an explicit End Program block with End selected in the fault handler—rather than just connecting the bottom (false) node of the last If Expression block to the END block of the module—if neither a global or runtime fault is active.

If multitasking was not enabled when the fault occurred and the End Program block with When End or Fault selected was executed, the program can be re-started via a Restart Program block after the fault is cleared, as shown below.

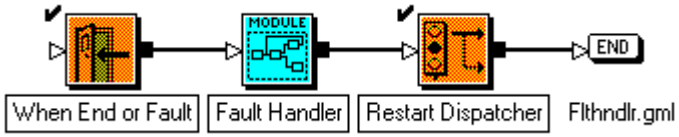


ATTENTION: Do not connect the output node of the Fault Handler module directly back to a previous block in the diagram.

If a global or runtime fault is still active when the Restart Program block executes, the program does not restart—the fault handler is immediately called again via the End Program block with When End or Fault selected. This ensures that all faults are cleared before the program restarts.

When restarting the program after executing the End Program block with When End or Fault selected, as shown above, it is imperative that you use an explicit Restart Program block—you must not directly connect the output node of the fault handler module back to a previous block in the diagram.

If multitasking was enabled when the fault occurred and the End Program block with When End or Fault selected executes, the program can also be re-started via a Restart Program block as explained previously. You can also use a Task Control block, with Restart Dispatcher selected, after the fault has been cleared to resume all tasks that were active when the fault occurred, as shown below.



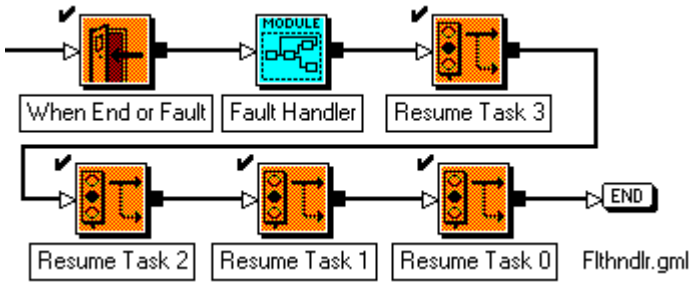
ATTENTION: Do not use a Task Control block, with Start New Task selected, in the fault handler

The Task Control block, with Restart Dispatcher selected, immediately turns multitasking ON again and re-activates all tasks (including the main task) which were active when the fault occurred. Each task continues from where it was interrupted when the fault occurred. Do not use a Task Control block with Start New Task selected to start or restart tasks in the fault handler.

If a global or runtime fault is still active when the Task Control block with Restart Dispatcher selected executes, the task dispatcher does not restart. The fault handler is immediately called again via the End Program block, with When End or Fault selected. This ensures that all faults are cleared before the program continues.

The connection from the Restart Dispatcher block to the END block of the diagram is never executed but is required to avoid a translation error due to an unconnected output node on the restart dispatcher block when the diagram is translated to a script.

If multitasking was enabled when the fault occurred and the End Program block with When End or Fault selected executed, the program tasks can also be re-started via individual Task Control blocks, with Resume Task selected, after the fault is cleared.



ATTENTION: Blocks following a Resume Task 0 block, in the When End or Fault sequence, are not executed.

When using this technique, it is important that task 0 (the main task) be the last task resumed. Once task 0 is resumed, the When End or Fault sequence is aborted and any blocks following a Resume Task 0 block are not executed.

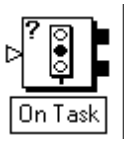
As with any Task Control block, with Resume Task selected, each task continues from where it was interrupted when the fault occurred. Do not use Task Control blocks, with Start New Task selected, to start or restart tasks in the fault handler.

If a global or runtime fault is still active when a Task Control block with Resume Task selected executes, the task dispatcher does not restart and the task does not resume—the fault handler is immediately called again via the End Program block with When End or Fault selected. This ensures that all faults are cleared before any task resumes. The connection from the Resume Task 0 block to the END block of the diagram never executes, but is required to avoid a translation error due to an unconnected output node on the task control block, with resume task selected, when the diagram is translated to a script.

Multitasking Blocks

Multitasking blocks control the starting and stopping of multiple independent tasks (multitasking) in the motion controller. The motion controllers allow execution of up to ten separate tasks simultaneously.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	• Turn on multitasking and begin executing a new task (Start New Task Type) • Stop execution of the current task (Stop Current Task Type) • Stop execution of a selected task, other than the current task (Stop Other Task Type) • Resume execution of a previously stopped task (Resume Task Type) • Turn off multitasking by disabling the task dispatcher (Stop Dispatcher Type) • Turn on multitasking by re-enabling the previously disabled task dispatcher (Restart Dispatcher Type)
	• Check the task dispatcher to determine whether or not a selected task is running • Pause the program until the selected task is running

Multitasking Operation

In the motion controllers, multitasking operates by means of a software device, called a task dispatcher, that constantly shifts its focus among all active tasks. If two tasks are active, the task dispatcher executes a block from the first task, then executes a block from the second task, then returns to execute the next block in the first task, and so on in alternation. If more than two tasks are active, the task dispatcher executes one block from each task in round robin fashion. Consequently, although most blocks do not hang the task dispatcher, the greater the number of active tasks, the slower each one runs. This type of operation is known as cooperative multitasking.

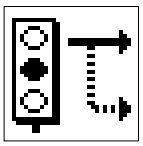
With multitasking, each task can be thought of as a separate program or sub-program operating concurrently with the other tasks. When multitasking is on, Task 0 is the main program as begun at the 

block at the beginning of the diagram. However, the 

block does not enable multitasking.

Within a task, conditional blocks and Wait for… blocks are evaluated once, then control returns to the task dispatcher. In this way, a Wait for *nn* or conditional block in one task does not pause execution of any other tasks or hang the task dispatcher.

Task Control



The Task Control Block is used to:

- Turn on multitasking and begin executing a new task (Start New Task Type).
- Stop execution of the current task (Stop Current Task Type).
- Stop execution of a selected task, other than the current task (Stop Other Task Type).
- Resume execution of a previously stopped task (Resume Task Type).
- Turn off multitasking by disabling the task dispatcher (Stop Dispatcher Type).

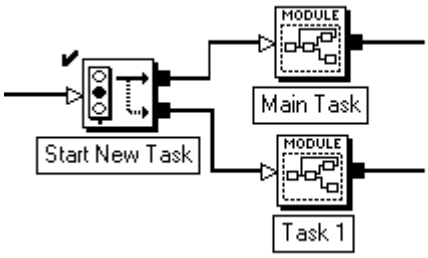
- Turn on multitasking by re-enabling the previously disabled task dispatcher (Restart Dispatcher Type).

The task control block resides on the Main Palette.

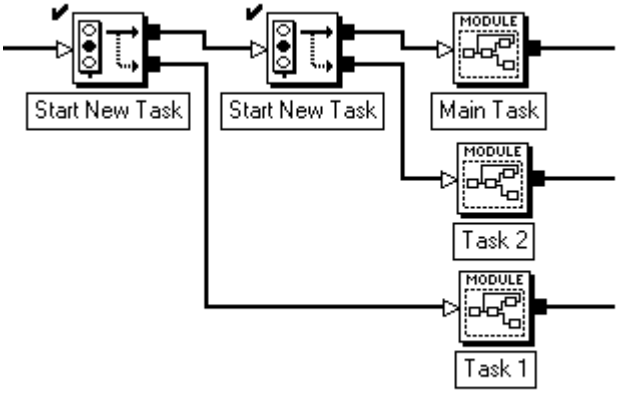
Start New Task

Start New Task enables multitasking and begins execution of a new task while execution of the current task continues. The new task blocks are attached to the bottom node, and the remaining current task blocks are attached to the top node. The two tasks continue operating concurrently (in parallel) until the program ends, or encounters a Task Control block with either Stop Current Task or Stop Other Task selected.

For example, the following GML Commander diagram starts a second task (task 1) while the main program (task 0) continues.

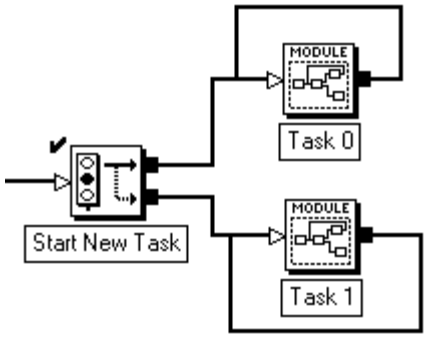


To start more than one additional task, cascade multiple Task Control blocks with Start New Task selected.

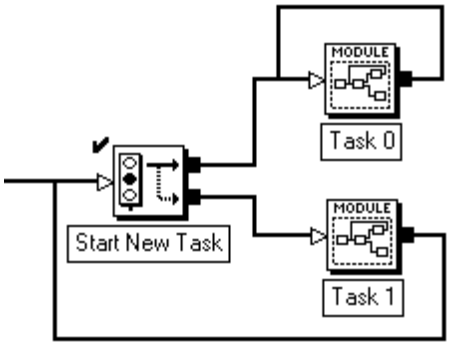


When starting multiple tasks, make sure that each task has a unique task number. Task numbers should not be re-used until the current task using that number is stopped (see *Stop Current Task* and *Stop Other Task*). Otherwise, unexpected program execution and machine operation may occur.

Sometimes it is necessary to loop program flow to the beginning of a task. When you do this, be sure to make your connections at the correct point in your program. For example, the following diagram loops program flow for each of two tasks back to the beginning of that task:



By contrast, the following diagram presents an example of incorrect looping, and causes a runtime error that ends program execution.



Stop Current Task

Stop Current Task stops execution of the task with the Task Control block and Stop Current Task selected.

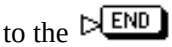
Think of each task as a separate concurrent program. Thus, stopping a task is equivalent to reaching the END block in a diagram, when multitasking is not enabled. If a task has been stopped via a Task Control block with Stop Current Task selected, it cannot be resumed by a Task Control block with Resume Task selected.

The Task Control block, with Stop Current Task selected, is like the END block, in that it has no output node.

ATTENTION: Do not stop all currently executing tasks. This may cause unexpected behavior which could cause damage to machinery, application, or personal injury.

Stopping all currently executing tasks aborts the diagram with a No Tasks Running runtime fault (Runtime_fault = 23).

Stop Other Task

Stop Other Task stops execution of the selected task. As each task can be thought of as a separate concurrent program, stopping a task is equivalent to the 

block in the diagram when multitasking is not enabled. However, unlike the end of a diagram, a stopped task can be resumed. See *Resume Task*, below, for more information on resuming tasks.

After you stop a task, you can re-use the task number for another task. However, once the task number has been re-used, the original task cannot be resumed.

Stopping all currently executing tasks aborts the diagram with a No Tasks Running runtime fault (Runtime_fault = 23).

Resume Task

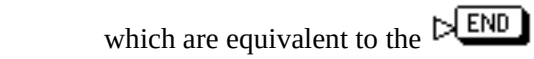
Resume Task resumes execution of a previously stopped task. The Task Control block, with Resume Task selected, has one output node.

ATTENTION: Do not resume a task that has not been previously started via a Start New Task block

To be resumed, a task must have been stopped by a previous Task Control block with Stop Other Task selected, or by executing a When End or Fault command in an End Program block after a global or runtime fault. The selected task continues from where it was when it was stopped. See *When End or Fault* in the *End Program* section in the *Program Control Blocks* chapter for more information on resuming tasks after a fault.

Stop Dispatcher

Stop Dispatcher turns multitasking off by disabling the task dispatcher. This pauses all tasks except the one that contains the Task Control block with Stop Dispatcher selected. See *Multitasking Operation* in this section for more information on the task dispatcher. The Task Control block with Stop Dispatcher selected has one output node.

Unlike the Stop Current Task and Stop Other Task type commands—which are equivalent to the 

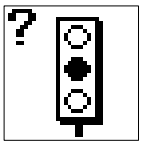
block in a diagram without multitasking—the Stop Dispatcher command suspends execution of all other tasks. It is sometimes useful to suspend multitasking briefly to accomplish time critical functions or calculations as quickly as possible, or to ensure full control over the operator interface so that multiple tasks do not interfere with each other.

After executing a Task Control block with Stop Dispatcher selected, the task containing this block is the only active task. You can resume the suspended tasks by turning multitasking ON again using a Task Control block with Restart Dispatcher selected.

Restart Dispatcher

Restart Dispatcher turns multitasking ON again by re-enabling the task dispatcher. This re-activates all tasks suspended by a previous Task Control block with Stop Dispatcher selected. Each task continues from where it left off when the task dispatcher was stopped.

On Task



The On Task block is a conditional block. Use the On Task block to:

- Check the task dispatcher to determine whether or not a selected task is running
- Pause the program until the selected task is running

The On Task block resides on the Main Palette.

If Task

If Task checks the task dispatcher to determine whether or not the selected task is running. The On Task block, with If Task selected, has two output nodes. Program flow branches to the top (true) node if the selected task is running, and to the bottom (false) node if not.

A task is not running if it has:

- not been started

- been suspended by a Task Control block with Stop Dispatcher type selected
- been stopped by a Task Control block with either Stop Current Task type or Stop Other Task type selected

Wait for Task

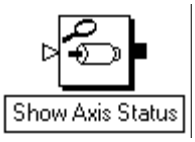
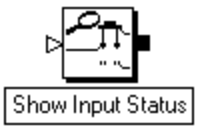
Wait For Task pauses the program until the selected task is running.

If other tasks are executing (multitasking), the task which contains this block pauses, but the other tasks continue to execute. In this way, an On Task block with Wait for Task selected in one task does not pause execution of any other tasks or hang the task dispatcher.

Status Blocks

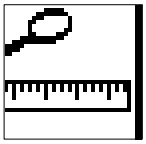
Status blocks report the status of the motion controller to the operator interface serial port. Blocks for reading axis position, axis status, discrete input status and controller memory status are included in this section.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Send the current position of the selected axis to the operator interface serial port.
	Send the current status of the selected axis to the operator interface serial port.
	Send the current state of a selected input to the operator interface serial port.
	Send to the operator interface serial port: • The current status of the application program or of a selected task, or the current value of the Runtime_fault variable • The motion controller's operating system firmware number • Information about the integrity of the stored application program or of the power-up values of the setup parameters, or the size of the current application program • The application identification string of the current program in the motion controller

When you select DH-485 Link in the Configure Control Options dialog box, serial port B on the motion controller is used for DH-485 communication and the normal built-in operator interface functions of GML Commander and the motion controllers are unavailable. The Status blocks may still be used for troubleshooting, however, if you select serial port A for the operator interface. See *Configuring Your Axis* chapter in this manual and the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on selecting the operator interface serial port.

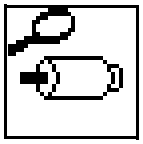
Show Axis Position



Use the Show Axis Position block to send the selected current position type of the selected axis to the operator interface serial port.

The Show Axis Position block resides on the Main Palette.

Show Axis Status



Use the Show Axis Status block to send the current status of the selected axis to the operator interface serial port.

The Show Axis Status block resides on the Main Palette.

The Show Axis Status block sends the current status of the selected axis to the operator interface serial port. See *Configuring Your Axis* in this manual for information on configuring axes.

GML Commander displays the current status of the selected axis as a numerical status value according to the following table. The table also shows the corresponding message that appears in an axis status field in the runtime display (if enabled). See *Configure Auto Display* in this manual for information on defining the runtime display.

Axis Status Values

Status Value	Description	Runtime Display	S	MO	V	I
14	AxisLink Timeout	AXL FLT			✔	
13	AxisLink Failed (axis not fount)	AXL FLT			✔	
12	Virtual Axis Not Enabled	AXS OFF			✔	
11	Drive Fault	DRV FLT	✔			
10	Position Error Tolerance Fault	ERR FLT	✔			
9	Hardware Overtravel Fault	HRD LIM	✔			
8	Software Travel Limits Fault	SFT LIM	✔			
7	Encoder Noise or Loss Fault	ENC FLT	✔	✔		
6	Feedback OFF or Virtual Axis Enabled	SRV OFF SRV OFF	✔	✔	 ✔	
5	Servo Output Limited	OUT LIM	✔			
4	Homing	HOMING	✔	✔	✔	
3	Moving or Executing Time-Lock Cam	 MOVING	 ✔			 ✔
2	Jogging	JOGGING	✔			✔
1	Axis Unlocked	UNLOCK	✔			✔
0	Axis Locked	LOCKED	✔			

In the above table, the columns at the right show whether each status value is valid for servo (S), master only (MO), virtual (V), or imaginary (I) axes.

Axis status conditions are prioritized from highest to lowest as shown in the previous table. The higher the status value, the higher its priority or severity. For example, a hardware overtravel fault has higher priority than a servo output limited condition. When a given status is active, other status conditions of lower priority can also be active. If a hardware overtravel fault is active for an axis, a servo output limited or feedback off condition can also be active on the axis. When the highest priority status is no longer active, the next highest priority active status value is reported by a subsequent Show Axis Status block.

Show Input Status

Use the Show Input Status block to send the current state of a selected input to the operator interface serial port.

The Show Input Status block resides on the Main Palette.

The Show Input Status block sends the current state of the selected Dedicated, Configured or Miscellaneous input to the operator interface serial port. The input status is 1 if the input is on, and 0 if it is off.

The status displayed is the current physical state (on or off) of the input. The Contacts setting (normal open or normal closed) in the motion controller's machine setup menu is ignored for the overtravel and drive fault inputs.

Select the **Input Class** from the opening page of the Show Input Status block. A corresponding tab displays for the type selected. Click on the tab to open the page for configuring the selected Input Class.

Dedicated

To configure this page, make entries in the following fields:

Axis – Select the physical axis, for which a dedicated input will be read.

Input – Select the dedicated axis input, the status of which (ON or OFF) will be read.

The available selections depend upon the Controller Type selected in the General page of the Configure Control Options dialog box and the Axis selected, above.

The status displayed is the current physical state (ON or OFF) of the input. The settings (Normally Open or Normally Closed) for Hard Travel Limits and Drive Fault Input (set in the Overtravel and Servo pages of the Configure Axis use dialog box) are ignored.

Configured

To configure this page, make entries in the following fields:

Tag Explorer – Select an input type from the tree control. The specific inputs or input bits associated with the selected input type appear in the Tag Window.

Note: If an input type does not appear, it has not been enabled.

Tag Window – Select the discrete input, the status of which (ON or OFF) will be read

Note: If a specific input does not appear, it has not been enabled.

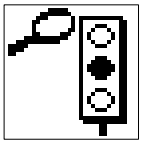
Miscellaneous

To configure this page, select a miscellaneous input from the menu.

The available inputs vary depending upon the controller type and firmware version you are operating.

Inputs with 1394 appended to their name are available only for the IMC S Class 1394/1394 Turbo servo controller.

Show Program Status



The Show Program Status block sends to the operator interface serial port:

- The current status of the application program or of a selected task, or the current value of the Runtime_fault variable (Show Program Status Type).
- The motion controller's operating system firmware number (Show Firmware ID Type).
- Information about the integrity of the stored application program or of the power-up values of the setup parameters, or the size of the current application program (Show Memory Status Type).
- The application identification string of the current program in the motion controller (Show Application ID Type).

The Show Program Status block resides on the Main Palette.

The four Show Program Status options from the For menu are described below.

Program Status

The Program from the For menu displays the execution status of the application program at the operator interface as a value. The following table displays the Status Values and their descriptions.

Program Status Values

Status Value:	Description:
2	Application Program Paused
1	Application Program Running
0	Application Program Stopped

Specific Task Status

The Specific Task option displays the execution status of the task selected Task Number menu at the operator interface. The following table shows the Task Values and their descriptions.

Program Task Values

Task Value:	Description:
2	Task Suspended
1	Task Running
0	Task Stopped or Never Started

A task is running after it has been started or resumed by a Task Control block, with either Start New Task or Resume Task selected. A task is not running if it has not been started or has been stopped by a Task Control block, with Stop Current Task or Stop Other Task selected. A task is suspended if it has been paused by a Task Control block, with Stop Dispatcher selected.

Runtime Fault Status

The Runtime Fault option displays the current value of the Runtime_fault system variable at the operator interface . The value displayed indicates the specific runtime fault that occurred.

Show Firmware ID

The Show Firmware ID option sends the motion controller's operating system firmware number to the operator interface serial port.

Use the Show Firmware ID type to identify the current operating system firmware in the motion controller when you are debugging a GML Commander diagram. You must refer to this firmware number when you report a problem, and to identify whether a motion controller is using the most recent firmware.

The firmware number contains eight characters, formatted as shown in the table below.

Firmware Numbering System: AAABBC.DE

Field	Description	Value
AAA	Product Number 1394 1394 Turbo Compact S Class Integrated/Basic	 295 308 303 238
BB	Firmware Configuration	00 or 10
C.D	Firmware Version	3.0
E	Firmware Revision	A-Z

Show Memory Status

The Show Memory Status type checks the integrity of the stored application program, the integrity of the power-up values of the setup parameters, or the size of the current application program and sends the resulting status to the operator interface serial port.

The motion controllers use a checksum to verify the integrity of memory. Whenever the contents of memory are changed through normal, proper methods (downloading an application program, executing the motion controller's setup menus, etc.), a new checksum for the appropriate memory area is calculated and stored with the data.

For Setup Data and Application Program checking, the Memory Status type re-calculates the checksum and compares it with the previously stored value to determine whether memory has been corrupted. Memory status is shown as 0 if the information stored in memory is valid (intact), and as 1 if the information has been corrupted.

For Program Size checking, the Memory Status type merely displays the size of the program.

Setup Data

If a Setup Data Memory Status type displays a 1—indicating corruption of the setup parameters—verify that all setup parameter values are correct by reviewing the data within the Configure Control Options and Configure Axis Use dialog boxes for each enabled axis. Then, with Download Axis/Drive Data enabled within the Configure Control Options dialog box, re-download the application program.

ATTENTION: After re-downloading, be sure to lock the memory using the front panel keyswitch or the memory unlock jumper to avoid future data corruption.

Application Program

If an Application Program Memory Status type displays a 1—indicating corruption of the application program—re-download the application program. After re-downloading, be sure to lock the memory using the front panel keyswitch or the memory unlock jumper to avoid future data corruption.

Program Size

When you select Program Size from the For menu, the operator interface displays the size of the current application program (generated from the GML Commander diagram and downloaded to the motion controller) in bytes. Controller maximum program sizes are as follows:

Controller Type:	Maximum Program Size:
1394	32 KB
1394 Turbo	64 KB
Compact	32 KB
S Class Integrated/Basic	32 KB

The Show Application ID type is most useful when debugging a GML Commander diagram to identify the current application program in the motion controller.

AxisLink Blocks

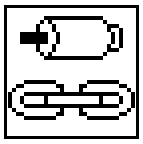
AxisLink blocks control the operation of AxisLink and are used only with motion controllers with AxisLink (1394-SJTxx-, 1394-SJTTxx, C-RL, and IMC-S/2xx-L models). Blocks for enabling and disabling virtual axes, reading user variables, data parameters, and data bits from other motion controllers on AxisLink, and clearing AxisLink faults are included in this section.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Enable and disable AxisLink virtual axes.
	Let one motion controller in an AxisLink chain request a user variable value, data parameter value, or data bit state from another motion controller in the chain.
	Directly clear any and all AxisLink axis specific faults on the selected virtual axis, by disabling the axis and setting to zero the value of the AxisLink fault variables for the axis.

Before you can use any AxisLink block, you must first be sure to enable AxisLink in the General and AxisLink pages of the Configure Control Options dialog box. See *Configuring Your Axis* in this manual for information on setting up AxisLink and Extended AxisLink.

Virtual Axis Control



The Virtual Axis Control block enables and disables AxisLink virtual axes. When you enable a virtual axis, you can use it as the master axis for electronic gearing and position-lock cams, just as if it were another physical master only axis. Only one of the two available virtual axes can be enabled at the same time—enabling one virtual axis automatically disables the other virtual axis.

The Virtual Axis Control block resides on the Advanced Motion Palette.

When a virtual axis is ON (enabled)

- AxisLink_status = 1
- Axis_status = 6
- Axis_fault = 0, if no faults are active on the axis
- Global_fault = 0, if no other faults are active on any axis

When a virtual axis is OFF (disabled)

- AxisLink_status = 0
- Axis_status = 12, if no faults are active on the axis
- Global_fault = 0, if no faults are active on any axis

The Axes System Variable AxisLink_failed indicates whether or not an attempted AxisLink connection was successful. It has a value of 1 (true) if an AxisLink virtual axis connection failed to be established (the axis could not be found), and 0 (false) if not. When AxisLink_failed = 1, Axis_fault = 6, Axis_status = 13, and Global_fault = 9.

Wait for Linked

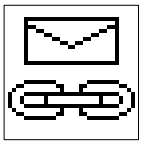
You can select Wait for Linked only if you first specify the On state.

Selecting Wait for Linked halts execution of the program until the virtual axis is in the selected On state. Because enabling a virtual axis can take up to 10 milliseconds, this guarantees the virtual axis is ready before it is used in subsequent blocks. In most cases, you should select Wait for Linked.

If other tasks are executing (multitasking), the task that contains this block pauses while the other tasks continue to execute. In this way, a Wait for Linked selection in one task does not halt execution of any other tasks or hang the task dispatcher.

When Wait for Linked is not selected, the program continues to execute all subsequent blocks while the virtual axis is being enabled or disabled.

Read Remote Value



The Read Remote Value block allows one motion controller in an AxisLink chain to request a user variable value, data parameter value, or data bit state from another motion controller in the chain. The local motion controller reads the selected item from the remote motion controller with the specified address, and stores the returned value in a selected local user variable. Typically, the value is returned in 3 to 5 ms, but may take as long as 20 ms. Both motion controllers must have the AxisLink option installed and enabled, and be connected on the same AxisLink chain.

The Read Remote Value dialog box's appearance differs depending on the type you select.

The Read Remote Variable block resides on the Advanced Motion Palette.

See the *User Variable Type* and *Data Type* for instructions on configuring the two versions of this function block.

The AxisLink_general_fault variable indicates whether or not the attempted AxisLink connection was successful. It has a value of 1 (true) if there has been an AxisLink timeout attempting to read the value from the remote motion controller and 0 (false) if not. When AxisLink_general_fault = 1, Global_fault = 8 if no other faults of higher priority are active on any axis.

When an Axislink_general_fault has a value of 1 (true) due to an AxisLink timeout attempting to read the value from the remote motion controller the AxisLink_fault_code = 64-79.

User Variable Type

When you select User Variable from the Type menu, the local motion controller reads a value from the selected remote motion controller's selected user variable number, and stores that value locally in a selected user variable.

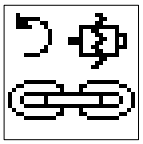
Data Type

When you select Data from the Type menu, the local motion controller reads either the current working state of the data bit, or the current working value of the data parameter, from the selected remote motion controller, and stores that value locally in a selected user variable.

Data Bits and Data Parameters

The available data bits and data parameters are displayed in two separate tables in *Miscellaneous Blocks* chapter and explained in the *Setup* section of the *Installation and Setup* manual for your motion controller.

Reset AxisLink Fault



The Reset AxisLink Fault block directly clears any and all AxisLink axis specific faults on the selected virtual axis by disabling the axis and setting the value of the AxisLink fault variables for the axis to zero.

The Reset AxisLink Fault block resides on the Advanced Motion Palette.

The Reset AxisLink Fault block sets the AxisLink_timeout, AxisLink_failed, and AxisLink_general_fault variables to zero. The Reset AxisLink Fault block only removes the fault status, it does not perform any other recovery such as re-enabling the virtual axis.

You should handle AxisLink faults as part of a global fault handling task or routine in the diagram. When an AxisLink fault occurs, the source of the problem must be corrected before clearing the fault with a Clear AxisLink Fault block and re-enabling the virtual axis, otherwise the fault re-occurs. An AxisLink fault in one motion controller can also cause AxisLink faults in other motion controllers in the chain, therefore, the fault recovery routine must consider the application programs running in other motion controllers. See *Handling Faults* in the *Using AxisLink* chapter of this manual for more information on handling AxisLink faults.

RIO Blocks

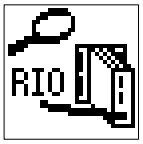
Use RIO blocks, together with an Allen-Bradley PLC, to check RIO status, control block transfers, and automatically update formatted data groups via RIO.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Check the RIO interface and send the resulting status value to the operator interface serial port.
	• Pause the current task until the selected RIO status condition is true. • Evaluate the status of RIO communications. • Pause the current task until the motion controller detects either a block transfer read (BTR) or write (BTW) from the RIO scanner. • Evaluate whether motion controller detects a block transfer read or block transfer write from the RIO scanner
	• Initiate automatic updating to selected RIO data fields. • Cancel automatic updating to selected RIO data fields.
	• Reset the RIO option after a communications failure.

Before you use RIO, be sure to first enable the RIO Interface in the General and RIO pages of the Configure Control Options dialog box.

Show RIO Status



Use the Show RIO Status block to check the RIO interface and send the resulting status value to the operator interface serial port.

The Show RIO Status block resides on the RIO Toolbar.

The Show RIO Status block requires no parameters, and has no dialog box.

If you enabled RIO Interface in the General and RIO pages of the Configure Control Options dialog box, you can use the Show RIO Status block to check the RIO interface and send the its status value to the operator interface serial port. This block requires no parameters and it always displays a check mark by the block's upper left corner to indicate that it is complete.

See the *Using the RIO Adapter Option* chapter in this manual for information on configuring RIO operation.

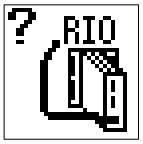
RIO status appears as a value according to the following table. You can also determine the RIO status by the color of the LED.

RIO Status Values

Value	Description	LED Color
4	Non-Recoverable Fault	Red
3	Recoverable Error (Failing)	Blinking Red
2	Standby	Blinking Green
1	Offline	Off
0	Online	Green

See the section *On RIO* in this manual for more information on the specific status conditions.

On RIO



Use the On RIO block to:

- Pause the current task until the selected RIO status condition is true.
- Evaluate the status of RIO communications.
- Pause the current task until the motion controller detects either a block transfer read (BTR) or write (BTW) from the RIO scanner.

Evaluate whether motion controller detects a block transfer read or block transfer write from the RIO scanner

The On RIO block is on the RIO Toolbar.

See *Using the RIO Adapter Option* chapter in this manual for information on configuring RIO operation.

Wait for RIO Status

Use the On RIO block, with Wait for RIO Status selected, to check for a specific RIO condition or fault. If other tasks are executing (multitasking), the task that contains this block pauses while the other tasks continue to execute. In this way, an On RIO block with Wait for RIO selected in one task does not pause execution of any other tasks or hang the task dispatcher. See *RIO Status Conditions* and the status descriptions, below, for information on each specific status condition.

If RIO Status

The program flow branches to the top (true) node if the current RIO status matches the selected status, and to the bottom (false) node if not. Use an On RIO block, with If RIO Status selected, to check for a specific RIO condition or fault.

RIO Status Conditions

You can also determine the RIO status from the color of the diagnostic LED for each channel on the front panel of the RIO option.

RIO Status Conditions

Description	LED Color
Non-Recoverable Fault	Red
Recoverable Error (Failing)	Blinking Red
Standby	Blinking Green
Offline	Off
Online	Green

The Remote I/O Status Conditions are explained below.

Non-Recoverable Fault

When you select Non-Recoverable Fault from the RIO Status menu, the current task pauses until (Wait for RIO Status), or branches to the top—true—node if (If RIO Status), one of the following faults occurs:

- Remote I/O link has failed due to excessive transmission errors
- Remote I/O link has become disconnected
- Scanner in the PLC has stopped scanning

Reset the Remote I/O Adapter using a Reset RIO block, or press the front panel RESET button to reset the motion controller. See the *Setup* section of the *Installation and Setup* manual for your motion controller and *Reset RIO* in this section.

Recoverable Error

When you select Recoverable Error from the RIO Status menu, the current task pauses until (Wait for RIO Status), or branches to the top—true—node if (If RIO Status), the Remote I/O link is failing due to transmission errors. This means that the RIO link is still operating, but fails completely in the future, at which point the RIO status changes to non-recoverable fault.

Standby

When you select Standby from the RIO Status menu, the current task pauses until (Wait for RIO Status), or branches to the top (true) node if (If RIO Status), the Remote I/O link is in standby mode. When standby mode concludes, the status changes to Online.

Offline

When you select Offline from the RIO Status menu, the current task pauses until (Wait for RIO Status), or branches to the top (true) node if (If RIO Status) one of the following occurs:

- there is a problem with the RIO link itself
- the RIO option in the motion controller is not functioning properly

After resetting the RIO adapter, its status is Offline while awaiting the first scan from the RIO scanner. After the adapter detects a proper scan from the scanner, its status changes to Online.

Online

When you select Online from the RIO Status menu, the current task pauses until (Wait for RIO Status), or branches to the top—true—node if (If RIO Status), the Remote I/O Adapter is communicating properly with the PLC. This is the normal status condition.

Wait for RIO Block Transfer

Use Wait for RIO Block Transfer to pause the program until the motion controller detects either a block transfer read (BTR) or write (BTW) from the RIO scanner.

On RIO blocks, with Wait for RIO Block Transfer selected, are not usually required unless block transfers to and from the PLC must be synchronized to the program in the motion controller (generated from the GML Commander diagram). More commonly, the motion controller program must be synchronized to the PLC program. In this case, it is easier to use discrete bits to synchronize and interlock block transfers in the PLC program.

If other tasks are executing (multitasking), the task that contains this block pauses while the other tasks continue to execute. In this way, a Wait for RIO Block Transfer selection in one task does not pause execution of any other tasks or hang the task dispatcher.

If RIO Block Transfer

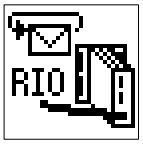
Use the If RIO Block Transfer selection to evaluate whether a block transfer read or write from the RIO scanner has occurred.

Program flow branches to the top (true) node if the selected type of block transfer has occurred, and to the bottom (false) node if not.

Use On RIO blocks, with If RIO Transfer selected, to check for an RIO fault. For both block writes and block reads, the condition is true only once immediately after the appropriate block transfer has been completed.

On RIO blocks, with If RIO Block Transfer selected, are not usually required unless block transfers to and from the PLC must be synchronized to the program in the motion controller (generated from the GML Commander diagram). More commonly, the motion controller program must be synchronized to the PLC program. In this case, it is easier to use discrete bits to synchronize and interlock block transfers in the PLC program.

Auto RIO Update



Use the Auto RIO Update block to:
- Initiate automatic updating to selected RIO data fields.
- Cancel automatic updating to selected RIO data fields.

You must make the necessary RIO configuration settings in the General and RIO pages of the Configure Control Options dialog box to ensure that all of this dialog box's required fields are enabled. For example, failure to select a sufficiently large rack size in the RIO page of the Configure Control Options dialog box disables the Auto RIO Update block's Type and List windows.

See *Using the RIO Adapter Option* chapter for information on configuring RIO operation.

When you select RIO interface in the General page of the Configure Control Options dialog box; the Auto RIO Update block with turn off selected, cancels the automatic updating of the selected RIO Adapter or Scanner Formatted Data Output Group as initiated by a previous Auto RIO Data Update block.

When you select RIO interface in the General page of the Configure Control Options dialog box, the Auto RIO Update block initiates automatic updating of the selected RIO Adapter or Scanner Formatted Output, with the value of the entered variable or expression. You use this block to automate the display of motion or status values on an Allen-Bradley PanelView Intelligent Operator Panel, or another operator interface device. Up to four such automatically updated data groups can be defined concurrently via four Auto RIO Data Update blocks.

The Display Refresh Time parameter in the motion controller's application setup menu determines the specified formatted data output group's update rate. See the *Installation and Setup* manual for your motion controller for information on setting the display refresh time parameter.

The specified formatted data output group receives the current value of the Output Data expression as a 16-bit binary or 7-digit BCD value along with a decimal point position and sign bit. The Allen-Bradley PanelView Intelligent Operator Panel automatically handles these formats for displaying values.

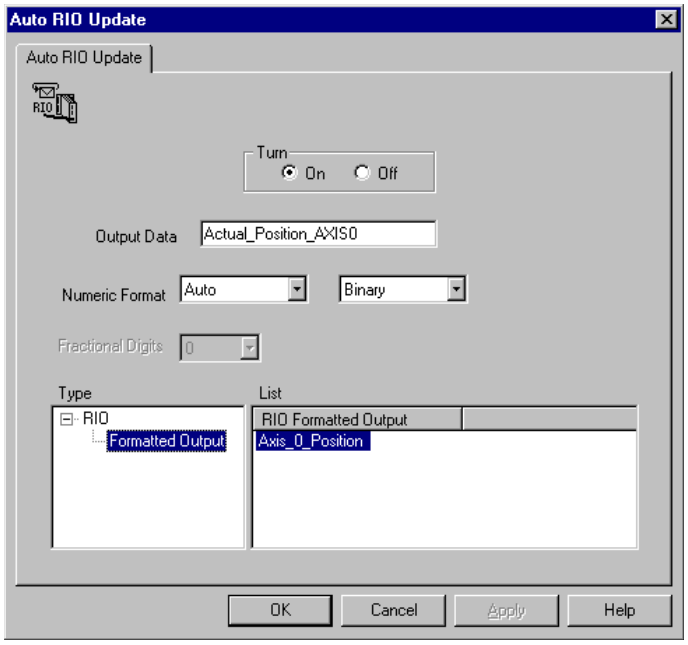
The Numeric Format menu combination selections are explained below.

Auto Binary Numeric Format

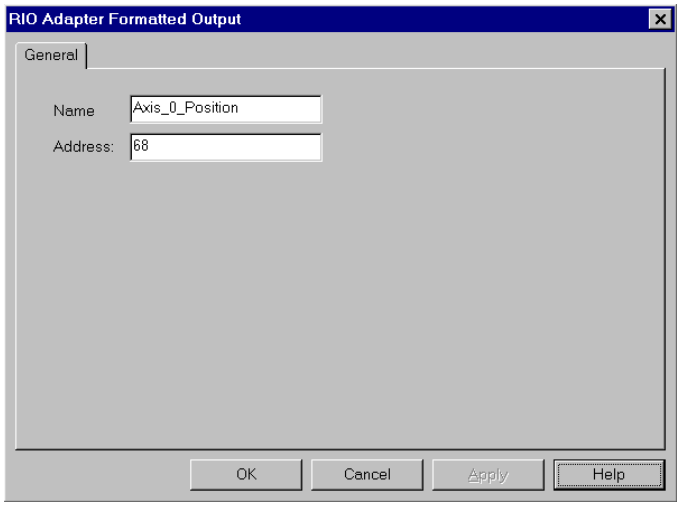
Select Auto from the first numeric format menu and Binary from the second menu to write the 16-bit binary value (magnitude only, not 2s complement) of the output data expression to the lower numbered I/O group defined for the selected formatted data output group.

The number of digits to the right of the decimal point (automatically determined by the motion controller to maintain the maximum number of significant digits in the displayed value) and the sign of the value are written to the four highest bits in the next higher numbered I/O group. The actual number of decimal digits changes from update to update as the value of the output data expression changes.

To continuously send the actual position of Axis 0 to a PLC-5 using a previously defined formatted data output group (Axis_0_Position), enter Actual_Position_AXIS0 in the *Output Data* field and select Axis_0_Position from the RIO Formatted Output List.



When using the RIO option as an adapter, and it is set to rack 4 using a full rack starting at I/O Group 0, the following definition for the RIO adapter formatted data output group results in the following input image table in the PLC for rack 4:



I/O Group	15 (SLC) 17 (PLC)							08 10	07 07							00 00
0	**Block Transfers**															
1	3	2	1	0	**Dedicated Discrete I/O**											
2	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
3	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
4	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36
5	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52
6	**Formatted Data Output Group Magnitude**															**68**
7	S	D2	D1	D0	95	94	93	92	91	90	89	88	87	86	85	84

The magnitude of the value is written to I/O Group 6 (RIO adapter user-defined discrete outputs 68 – 83).

The least significant bit is at:

PLC `I:046/00` (RIO output 68)

The most significant bit is at:

PLC `I:046/17` (RIO output 83)

The sign bit is:

PLC `I:047/17` (RIO output 99)

It is 0 for a positive value and 1 for a negative value.

PLC inputs `I:047/14` through `I:047/16` (RIO outputs 96 – 98) contain a value between 0 and 7 indicating the number of decimal digits (digits to the right of the decimal point) in the formatted data output group magnitude in I/O Group 6. The motion controller automatically determines this value to maintain the maximum number of significant digits in the displayed value. This value typically changes from update to update. Thus, if the current position of Axis 0 is –123.45, the PLC input image table for rack 4 is updated as shown below (x = don't care):

I/O	15							08	07				00			
Group	17		(SLC) (PLC)					10	07				00			
0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
4	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x
5	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1
7	1	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x

Value is negative `I:047/17` = 1
2 Decimal digits $2_{10} = 010_2 \Rightarrow$ `I:047/16, …/15,…/14`
Magnitude $123.45 \times 10^2 = 12{,}345 = 3039_{16} = 011\ 0000\ 0011\ 1001_2$

Fixed Binary Numeric Format

When you select Fixed from the first Numeric Format menu and Binary from the second, the values written to the formatted data output group are the same as with auto binary numeric format except that the number of decimal digits (digits to the right of the decimal point) is always the same. Select the desired number of decimal digits from the Fractional Digits menu. Fixed format gives a visually more stable display at the expense of absolute precision. See *Auto Binary Numeric Format* for details of the values written to the selected formatted data output group.

Because a 16-bit magnitude is written, the maximum value that can be represented is ±65,535. This limits the largest value that the motion controller can write based on the number of specified decimal digits as shown in the table below.

Binary Formatted Data Output Group Maximum Values

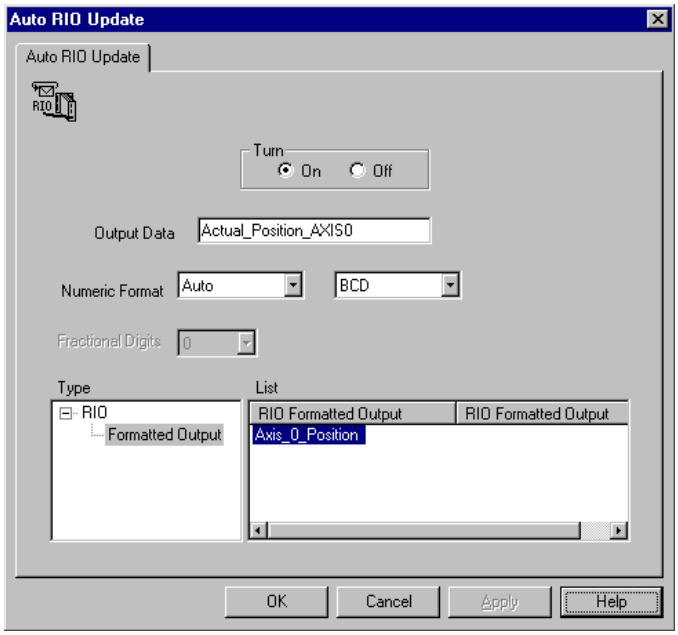
Number of Decimal Digits	Maximum Value
0	±65,535.0
1	±6,553.5
2	±655.35
3	±65.535
4	±6.553 5
5	±0.655 35
6	±0.065 535
7	±0.006 553 5

Be careful to select the number of decimal digits so that the expected maximum values can be represented with the required precision.

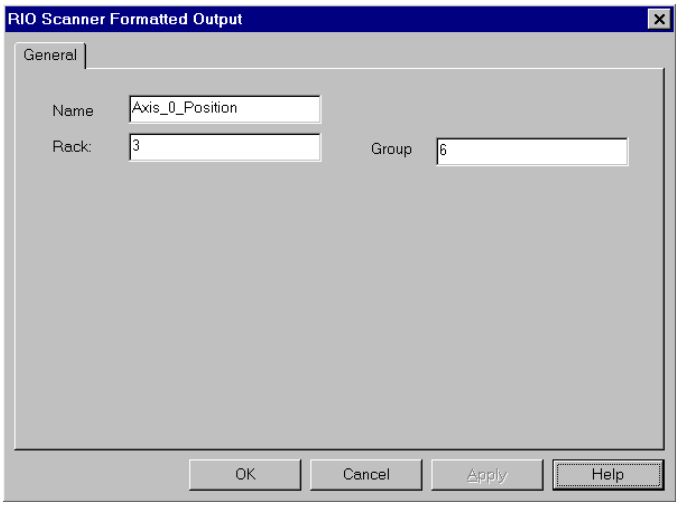
Auto BCD Numeric Format

When you select Auto from the first Numeric Format menu and BCD from the second, the lower four digits of the 7-digit BCD value of the output data expression are written to the lower numbered I/O group defined for the selected formatted data output group. The remaining three digits of the value, the number of decimal digits (automatically determined by the motion controller to maintain the maximum number of significant digits in the displayed value), and the sign of the value are written to the next higher numbered I/O Group. This is the data format used by the PanelView. The actual number of decimal digits changes from update to update as the value of the Output Data expression changes. This gives the highest precision over a large range of values.

To continuously send the actual position of Axis 0 to a PanelView using a previously defined formatted data output group (Axis_0_Position), enter Actual_Position_AXIS0 in the *Output Data* field and select Axis_0_Position from the Send to RIO Data Group list.



When using the RIO option as a scanner and the PanelView is set up as rack 3, the following definition for the RIO scanner formatted data output group results in the following output image table in the motion controller for rack 3.



I/O Group	15 (SLC) 17 (PLC)	08 10	07 07	00 00
0				
1				
2				
3				
4				
5				
6	BCD Digit 4	BCD Digit 3	BCD Digit 2	BCD Digit 1
7	S D2 D1 D0	BCD Digit 7	BCD Digit 6	BCD Digit 5

The four least significant digits of the magnitude are written to I/O group 6. The remaining three BCD digits are written to the lower 12 bits in I/O group 7. The sign bit is I/O group 7 bit 17 (PLC-5) or bit 15 (SLC). This bit is 0, if the value is positive and 1, if it is negative.

Bits 14 – 16 (PLC-5) or 12 – 14 (SLC) indicate the number of decimal digits (digits to the right of the decimal point) in the BCD magnitude. The motion controller automatically determines this value to maintain the maximum number of significant digits in the displayed value. This value typically changes from update to update. Thus, if the current position of Axis 0 is –123.45, the output image table for rack 3 is updated as shown below (x = don't care):

I/O Group	15 17		(SLC) (PLC)				08 10		07 07				00 00			
0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
4	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x
5	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1
7	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1

Value is negative `O:037/17 = 1`

2 Decimal digits $2_{10} = 010_2 \Rightarrow$ `O:037/16, .../15, .../14`

Magnitude $123.45 \times 10^2 = 0012345$

I/O Group 6 $2345_{10} = 0010\ 0011\ 0100\ 0101_{BCD}$

I/O Group 7 lower 12 bits $001_{10} = 0000\ 0000\ 0001_{BCD}$

Fixed BCD Numeric Format

When you select Fixed from the first Numeric Format menu and BCD from the second, the values written to the formatted data output group are the same as with auto BCD numeric format except that the number of decimal digits (digits to the right of the decimal point) is always the same. Select the desired number of decimal digits from the Fractional Digits menu. Fixed format gives a visually more stable display at the expense of absolute precision. See *Auto BCD Numeric Format* for details of the values written to the selected formatted data output group.

Because it writes to a 7-digit BCD magnitude, the maximum value that it represents is ±9,999,999. This limits the largest value that the motion controller can write based on the number of specified decimal digits as shown in the table below.

Binary Formatted Data Output Group Maximum Values

Number of Decimal Digits	Maximum Value
0	±9,999,999.0
1	±999,999.9
2	±99,999.99
3	±9,999.999
4	±999.999 9
5	±99.999 99
6	±9.999 999
7	±0.999 999 9

Be sure to choose the number of decimal digits that best represents the expected maximum values with the required precision.

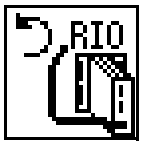
Tag Explorer

RIO Formatted Output is pre-selected.

Tag Window

Select the discrete output that receives (**On**) or no longer receives (**Off**) data.

Reset RIO



Use the Reset RIO block to reset the RIO option after a communications failure.

The Reset RIO block resides on the RIO Toolbar.

The Show RIO Status block requires no parameters, and no dialog box is associated with it.

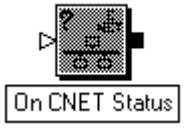
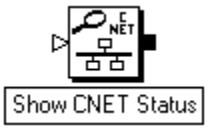
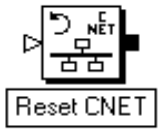
If you enabled RIO Interface in the General and RIO pages of the Configure Control Options dialog box, you can use the Reset RIO block to reset the RIO option after a communications failure. This block requires no parameters and it always has a check mark by the upper left corner of the block to indicate that it is complete.

See *Using the RIO Adapter Option* chapter in this manual for information on configuring RIO operation.

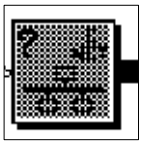
CNET Blocks

If you select CNET in the General page of the Configure Control Options dialog box, you can use CNET blocks to regulate communications between the motion controller and other devices via ControlNet (CNET).

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	• Pause execution of the current task at this block until CNET communications attains a specified CNET status condition. • Branch program flow from this block, depending upon whether a selected CNET status is true or false (If CNET Type).
	Use this block to check the ControlNet interface and send the resulting status value, or the location of the motion controller on the ControlNet network, to the operator interface serial port.
	Use the Reset CNET Fault block to clear all ControlNet faults and to reset the ControlNet card in the motion controller. This block sets the ControlNet global fault variable (Global_fault = 16) to zero.

On CNET Status



If you enabled CNET in the Configure Control Options General page dialog box, you can use the On CNET Status block to:

- Pause execution of the current task at this block until CNET communications attains a specified CNET status condition (Wait for CNET Status Type).
- Branch program flow from this block, depending upon whether a selected CNET status is true or false (If CNET Type).

The On CNET Status block is on the CNET Palette.

Wait for CNET Status Type

Select Wait for CNET to pause execution of the application program until the occurrence of a CNET status condition. When you select Wait for CNET, the block has a single exit node.

Wait for CNET halts execution of the current task until the specified CNET status (Online, Offline, Failing, or CNET fault) is true. If other blocks are executing (multitasking), only the task that contains this block pauses, while other tasks continue to execute. In this way, a Wait for CNET type block in one task does not halt execution of any other tasks, or hang the task dispatcher.

Online

Online status (CNET_status = 0) indicates normal CNET operation. The LED for the channel actively being used for CNET communications is solid green.

Offline

Offline status (CNET_status = 1) can indicate a number of different conditions such as: no power, the CNET plug card is conducting a self test.

Failing

Failing status (CNET_status = 2) indicates that the CNET plug card is experiencing temporary errors.

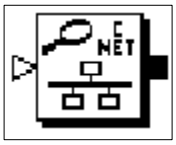
Fault

Fault status (CNET_status = 3) indicates that a CNET fault has occurred (i.e., CNET_fault = 1).

If CNET Status

Select **If CNET** to branch program depending upon the status of a specified CNET condition. The If CNET selection constantly checks the status of the CNET. Program flow branches to the true node (at the top of the block) if the current CNET status matches the status you specified in the block, and to the false node (at the bottom of the block) if not.

Show CNET Status



If you selected CNET in the Configure Control Options General page dialog box, you can use the Show CNET Status block to check the ControlNet interface and send the resulting status value, or the location of the motion controller on the ControlNet network, to the operator interface serial port.

The Show CNET Status block resides on the CNET Palette. To configure the Show CNET Status block you must make an entry in the Show field. You have two options:

- **Status** – to display the CNET status.
- **MAC ID** – to display the motion local controller's address (Media Access Control ID) on the ControlNet network.

Status

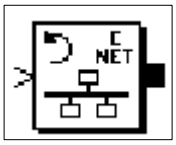
When you select Status from the Show menu, the ControlNet status appears as a value, as follows:

Value	Description
3	CNET Fault
2	Failing
1	Offline
0	Online

MAC ID

When you select MAC ID from the Show menu, the address (or Media Access Control ID) of the local controller appears as an integer value, from 1 to 99. The MAC ID value is set in the CNET page of the Configure Control Options dialog box.

Reset CNET



If you enabled CNET in the Configure Control Options General page dialog box, you can use the Reset CNET block to clear all ControlNet faults and to reset the ControlNet card in the motion controller. This block sets the ControlNet global fault variable (Global_fault = 16) to zero.

The Reset CNET block resides on the CNET Palette. To configure the Reset CNET Fault block, in the Type field select:

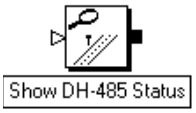
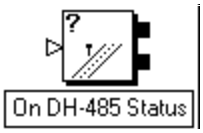
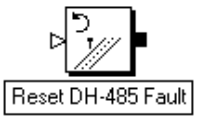
- **All Faults** – to clear the CNET_fault variable (Global_fault = 16) by resetting it to zero, or
- **Plug** – to reset the ControlNet plug card in the motion controller, thereby returning both the CNET_fault variable (Global_fault = 16) and the CNET_status variable to zero.

Use this block to reset CNET communications only after you have determined and fixed the cause of the fault or failure.

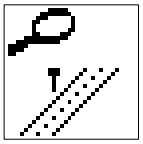
DH-485 Blocks

When DH-485 is selected in the General page of the Configure Control Options dialog box, you can use DH-485 blocks to transfer data between the motion controller and other devices via a DH-485 network.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Check the DH-485 interface and send the resulting status value, or the location of the motion controller on the DH-485 link, to the operator interface serial port.
	• Read a value (or multiple contiguous values) from a remote DH-485 element (or elements) and wait for the transfer to be acknowledged. • Send a value (or multiple contiguous values) from a local DH-485 variable in the motion controller to a remote DH-485 element (or elements), and wait for the transfer to be acknowledged.
	• Evaluate the status of DH-485 communications. • Pause the program until the specified DH-485 status condition is true.
	Clear any and all DH-485 faults

Show DH-485 Status



Use the Show DH-485 Status block to check the DH-485 interface and send the resulting status value, or the location of the motion controller on the DH-485 link, to the operator interface serial port.

The Show DH-485 Status block resides on the DH-485 Palette

If you select DH-485 in the General page of the Configure Control Options dialog box, you can use the Show DH-485 Status block to check the DH-485 interface and send the resulting status value, or the location of the motion controller on the DH-485 link, to the operator interface serial port.

Show Status

When you select Status from the Show menu, the DH-485 status appears as a value according to the following table.

DH-485 Status Values

Value	Description
2	Busy
1	Offline
0	Online

See *On DH-485* for more information on the specific status conditions.

Show Where Is

When you select Where Is from the Show menu, the node addresses of the motion controller and its adjacent neighbors on the DH-485 link are displayed.

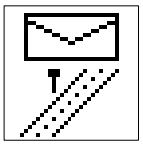
```
Pred:    MyAddr:    Succ:
```

- The motion controller’s node address is shown as My Addr.

- The node address of the motion controller's predecessor (the device which passes the token to the motion controller) is shown as Pred.
- The node address of the succeeding device (the one to which the motion controller passes the token) is shown as Succ.

When there is only one other device on the DH-485 network, the address of the predecessor is the same as that of the successor. If there are no other masters on the DH-485 network, all three node addresses are the same.

DH-485 Value



Use the DH-485 Value block to:

- Read a value (or multiple contiguous values) from a remote DH-485 element (or elements) and wait for the transfer to be acknowledged. (Read Type)
- Send a value (or multiple contiguous values) from a local DH-485 or CNET variable in the motion controller to a remote DH-485 element (or elements), and wait for the transfer to be acknowledged. (Send Type)

The DH-485 Value block resides on the DH-485 Palette.

For Send Type transfers, the black directional arrow points from the DH-485 or CNET Local variable list, on the left, to the DH-485 Remote variable list (or the Remote field group), on the right. For Read type transfers, the arrow's direction is reversed.

See the *Using the DH-485 Option* chapter for information on configuring DH-485 operation.

Read Type

When you select DH-485 in the Configure Control Options dialog box, the DH-485 Value block, with Read type selected, reads a value (or multiple contiguous values) from a remote DH-485 element and optionally waits until the requested data is received. The value or values read are stored in local DH-485 variables in the motion controller.

Send Type

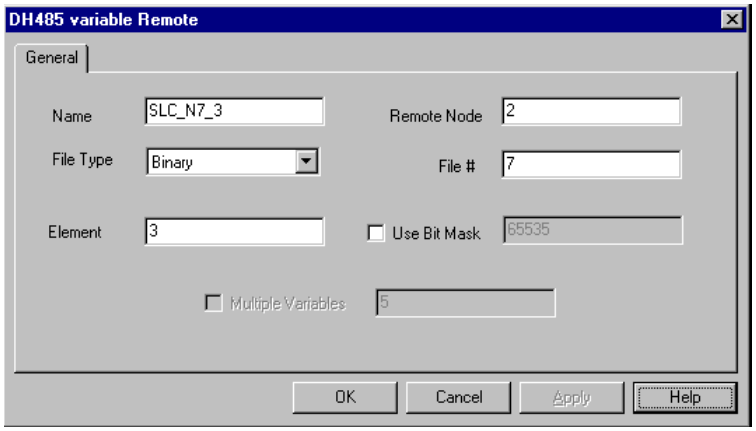
When you select DH-485 in the Configure Control Options dialog box, the DH-485 Value block, with Send type, it sends a value from a local DH-485 variable (or multiple contiguous values) in the motion controller, to a remote DH-485 element (or elements), and waits for the transfer to be acknowledged.

Specifying the Remote Element Indirectly

Not selecting Specify Remote Addressing, causes the DH-485 remote variable scrolling list to appear in the lower right part of the dialog box. The scrolling list contains previously defined remote elements. By selecting a previously defined remote element, you are specifying that remote element indirectly.

For example, the DH-485 Value block settings in the DH-485 Value dialog box displayed above, read the DH-485 remote element SLC_N7_3 (file element N7:3 in the SLC at node 2) into the DH-485 local variable Parts_Count.

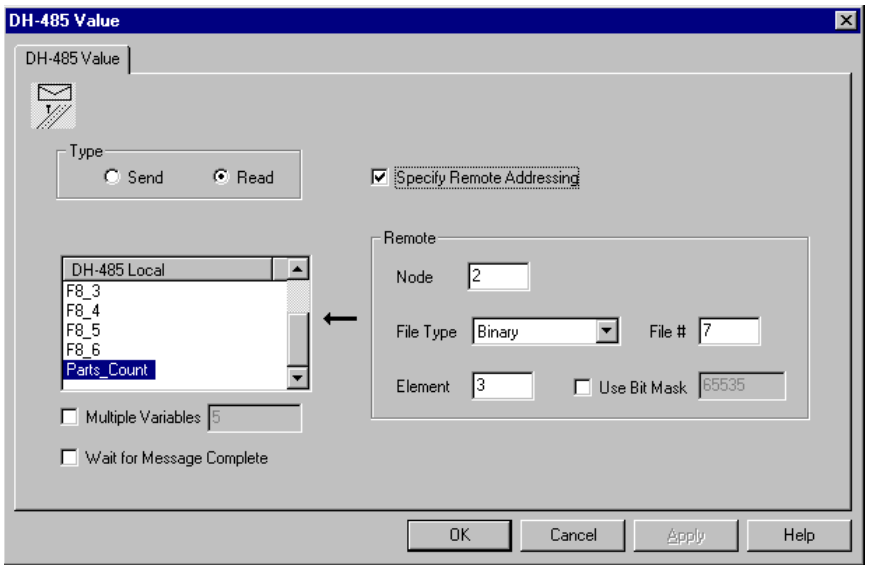
The remote element SLC_N7_3 was defined in the following dialog box accessed via the Tag Explorer and the Tag Window.



See the *Online Help* for more information on defining variables.

Specifying the Remote Element Directly

Selecting Specify Remote Addressing lets you directly specify the remote element, rather than as a previously defined item. The DH-485 Value dialog box replaces the DH-485 Remote scrolling list with a group of Remote fields to be completed. As shown below, these are the same fields that appear in the definition dialog box.



In the above example, the DH-485 Value block settings directly specify file element N7:3 in the SLC at node 2. These direct settings produce results identical to the indirect settings shown earlier.

Using Bit Masks

When you directly specify a remote element in a binary file, you can select the Bit Mask option. The bit mask is a decimal representation of a binary number that is bitwise “Anded” to the Binary DH-485 value.

Multiple Variables

The specified number of values—starting with the specified DH-485 remote element—is read from the remote device. The values are stored in an equal number of contiguous elements in a local DH-485 data file in the motion controller starting with the specified DH-485 or CNET local variable. Contiguous elements in the remote file are read, not contiguous items in the DH-485 Remote list. Likewise, the values read are stored in contiguous local elements, not contiguous items in the DH-485 Local list.

You do not have to define all the DH-485 remote elements—only the first. You do not have to define all the DH-485 local elements used to store the read values to successfully read multiple elements. However, any local elements whose values are required by the GML Commander diagram must be defined as DH-485 Local variables to allow access via the Equation block and the Expression Builder.

The maximum number of contiguous remote elements which may be read using a single DH-485 Value block depends on the file type selected for the element in question. See *DH-485 Message Details* for the limits for each file type.

The specified number of values (starting with the specified DH-485 local element) is sent to the remote device. The values are sent to an equal number of contiguous elements of the data file in the remote device, starting with the specified DH-485 remote element. Contiguous elements in the remote file are sent, not contiguous items in the DH-485 Local list. Likewise, the values are sent to contiguous remote elements, not contiguous items in the DH-485 Remote list.

You do not have to define all the DH-485 local variables—only the first. Nor do you have to define all the DH-485 remote elements used to store the values to successfully send multiple elements. However, any local elements whose values are required by the GML Commander diagram must be defined as DH-485 Local variables to allow access via the Equation block and the Expression Builder.

The maximum number of contiguous remote elements which may be sent using a single DH-485 Value block depends on the file type selected for the element in question. See *DH-485 Message Details* for the limits for each file type.

Wait for Message Complete

When you select Wait for Message Complete, one of the following occurs:

- the program pauses until the requested value is received and no DH-485 faults exist (in a Read type block)
- the remote device acknowledges receipt of the value (or values) and no DH-485 faults exist (in a Send type block)

When the requested value (read type) or acknowledgment (send type) is received, the program continues with the next block. If other tasks are executing (multitasking), the task that contains this block pauses but the other tasks continue to execute. A Wait for Message Complete selection in one task does not halt execution of any other tasks or hang the task dispatcher.

If you do not select Wait for Message Complete, the DH485 Value block just requests (read type) or sends (send type) the specified value or values and the program continues with the next block.

DH-485 Message Details

The DH-485 Value block, with Send Type selected, issues a Protected Typed Logical Write with Three Address Fields command on the DH-485 network. Similarly, the Read type block issues a Protected Typed Logical Read with Three Address Fields command on the DH-485 network.

Depending on the type and number of elements specified, up to 240 bytes of data may be included in the command packet. The number of data bytes in the command packet for each file type and the corresponding maximum number of elements, which may be specified in the *Multiple Elements* field, is shown in the following table.

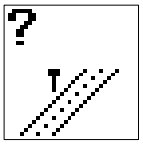
DH-485 Value Block
DH-485 Command Data Bytes

GML Commander File Type	Data Bytes per Element	Maximum Elements per GML Block
Binary	2 (16 bits)	120
Integer	2	120
Floating	4	60
ASCII	1	240
BCD	2	120
IntFloat	4	60

With binary files, each element contains 16 bits—or sub-elements—but the entire word containing the desired bit is transferred. A maximum of 1,920 bits may be represented in 240 data bytes.

See Allen-Bradley Data Highway/Data Highway Plus™/DH-485 Communication Protocol and Command Set Reference Manual (publication 1770-6.5.16), for more information on the Protected Typed Logical Write with Three Address Fields command.

On DH-485 Status



Use the On DH-485 Status block to:
- Evaluate the status of DH-485 communications.
- Pause the program until the specified DH-485 status condition is true.

The On DH-485 Status block resides on the DH-485 Toolbar.

See *Using the DH-485 Option* chapter in this manual for information on configuring DH-485 operation.

If DH-485

If you select DH-485 Interface in the General page of the Configure Control Options dialog box, the If DH-485 type block checks the status of DH-485 communications. Program flow branches to the top (true) node if the current DH-485 status matches the specified status, and to the bottom (false) node if not.

The DH-485 status conditions are explained in the following sections.

Online

Online is the normal status condition. When it is selected from the DH-485 Status menu, the program branches to the top (true) node if the motion controller is communicating properly with other devices on the DH-485 network.

Offline

When you select Offline from the DH-485 Status menu, the program branches to the top (true) node if the motion controller is not communicating with other devices on the DH-485 network. This condition can be caused by a problem with the DH-485 network itself, or a DH-485 fault in the motion controller. After clearing the DH-485 fault, the status changes to Online.

Busy

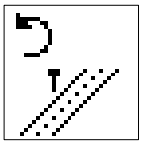
When you select Busy from the DH-485 Status menu, the program branches to the top (true) node if the motion controller is busy reading a value, sending a value, or waiting for an acknowledgment or reply from a remote device on the DH-485 network. See *DH-485 Value* for more information on reading and sending values on the DH-485 network.

Wait for DH-485

If you selected DH-485 in the General page of the Configure Control Options dialog box, the Wait for DH-485 type block halts the current task until the specified DH-485 status condition is true. Select the desired DH-485 Status wait condition from the pop-up menu. See *If DH-485 Status* for information on the each status condition.

If other tasks are executing (multitasking), the task that contains this block pauses while the other tasks continue to execute. In this way, a Wait for DH-485 selection in one task does not halt execution of any other tasks or hang the task dispatcher.

Reset DH-485 Fault



Use the Reset DH-485 block to clear any and all DH-485 faults.

The Reset DH-485 Fault block resides on the DH-485 Toolbar. It requires no parameters and has no dialog box. This block requires no parameters and it always has a check mark by the upper left corner of the block to indicate that it is complete.

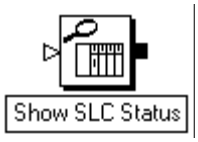
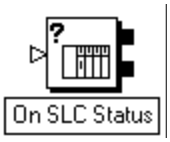
If you select DH-485 Link in the General page of the Configure Control Options dialog box, you can use the Reset DH-485 Fault block to directly clear any and all DH-485 faults. Use this block to reset DH-485 communications only after you have determined and fixed the cause of the fault or failure.

This block sets the DH485_general_fault variable to zero.

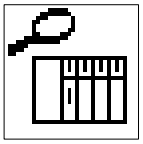
SLC Blocks

If you select SLC in the General page of the Configure Control Options dialog box, you can use SLC blocks to regulate communications between the motion controller and your Small Logic Controller (SLC).

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Display the status of the SLC.
	• Pause execution of the current task at this block until a the SLC attains a specified SLC status condition. • Branch program flow from this block, depending upon whether a selected SLC status is true or false.
	• Generate a user interrupt to the SLC.

Show SLC Status



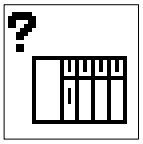
If you selected SLC in the General page of the Configure Control Options dialog box, you can use the Show SLC Status block to display the status of the SLC.

A 1394 Turbo motion controller is required for SLC operation.

The Show SLC Status block resides on the SLC Palette. It requires no parameters, and has no dialog box. This block requires no parameters and it always has a check mark by the upper left corner of the block to indicate that it is complete.

If you selected SLC in the General page of the Configure Control Options dialog box, you can use the Show SLC Status block to check the SLC interface and send the resulting status value to the operator interface serial port.

On SLC Status



If you enabled SLC in the General page of the Configure Control Options dialog box, you can use the On SLC Status block to:

- Pause execution of the current task at this block until the SLC attains a specified SLC status condition (Wait for SLC Status Type).
- Branch program flow from this block, depending upon whether a selected SLC status is true or false (If SLC Status Type).

The On SLC Status block is on the SLC Palette.

Wait for SLC Status Type

Use the Wait for SLC Status type block to check for an SLC fault.

The Wait for SLC Status halts execution of the current task until the specified SLC status, Programmed, Run, or Fault, (see below) is true. If other blocks are executing (multitasking), only the task that contains this block pauses, while other tasks continue to execute. A Wait for SLC type block in one task does not halt execution of any other tasks, or hang the task dispatcher.

When you select the Wait for SLC Status type, the block has a single exit node.

Program Status

Program status indicates the SLC is in the mode for creating, or programming, a ladder logic program in the SLC. When the SLC is in Program mode, it cannot be in Run mode or Fault mode.

Run Status

Run status indicates the SLC is executing, or running, a previously programmed SLC ladder logic program. When the SLC is in Run mode, it cannot be in Program mode or Fault mode.

Fault Status

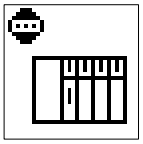
Fault status indicates that another module connected to the same backplane as the local 1394 system module (i.e., the SLC or another 1394 system module) has lost power. Refer to your SLC documentation for a description of these faults, and how to handle them.

If SLC Status

Like the Wait for SLC Status type block, use an If SLC Status type block to check for an SLC fault.

The If SLC Status type branches program flow, depending upon whether a selected SLC status is true or false. The If SLC Status type constantly checks the status of the SLC. Program flow branches to the true node (at the top of the block) if the current SLC status matches the status you specified in the block, and to the false node (at the bottom of the block) if not.

Interrupt SLC



If you enabled SLC in the General page of the Configure Control Options dialog box, you can use the Interrupt SLC block to generate a user interrupt to the SLC. A 1394 motion controller is required for SLC operation.

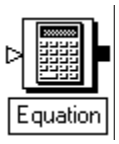
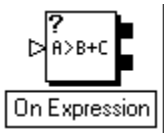
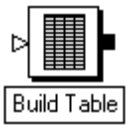
The 1394 GMC Turbo can issue a request for service command and generate an interrupt to the SLC processor. You can design an SLC ladder program to execute on receipt of the interrupt. Refer to the *APS Reference* manual for more information on SLC interrupts.

The Interrupt SLC block resides on the SLC Palette. It requires no parameters, and has no dialog box. Because this block requires no parameters, it always has a check mark by the upper left corner of the block, indicating that it is complete.

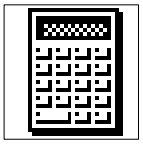
Calculation Blocks

Calculation blocks perform mathematical and logical calculations to define variable values, evaluate expressions, etc. Blocks for writing equations, evaluating mathematical or logical expressions, and building tables of values are included in this section.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Assign a value to: • a user variable • an indirect variable • certain system variables • a discrete output • an output group • a CNET local variable • a DH-485 local or remote variable • a cam profile point • an SLC variable • an SLC indirect variable.
	• Evaluate a mathematical expression and make a logical decision based on the expression's value. • Pause execution of a task or program until the value of a mathematical expression meets a specified criteria.
	Use the Build Table block to build time and position cam profiles, and arrays of user variables.

Equation



The Equation block assigns a value to:
- a user variable
- an indirect variable
- certain system variables
- a discrete output
- an output group
- a DH-485 variable
- a cam profile point
- an SLC variable
- an SLC indirect variable.

The assigned value can be a constant, or an expression that GML Commander evaluates to determine the assigned value.

The Equation block resides on the Main Palette.

The Equation block presents five different appearances depending upon your response in the Type box. The options include:

- Configured
- Slave_CAM_position
- Master_CAM_position
- Master_CAM_time
- Indirect_Variable

The steps for configuring each type of equation are set forth below.

Configured

Select Configured in the **Type** box to assign a value to an Axis System variable, an AxisLink I/O output or output group, a DH-485 variable, a General System variable, an SLC variable or a User variable. Make entries in the following fields:

- **Tag Explorer** – Select a variable type from the tree control. The specific variables associated with a the selected variable type appear in the Tag Window.

Note: If a variable type does not appear, it has not been enabled.

- **Tag Window** – Select a specific variable or output to hold the assigned value.

Note: If a particular variable or output does not appear, it has not been defined.

- = (value) – Enter the value or expression that will be assigned to the selected variable.

Master/Slave CAM Position types

Use the Master/Slave CAM Position type to assign a value to a Position Lock Cam Master or Slave Cam position. Make your entries in the following fields:

Axis – Select an axis.

Note: If an axis does not appear, it has not been enabled.

PCAM Slave Point – If you selected Slave_CAM_position, enter a value or expression for the PCam Slave Point number, from:·0 to 12,999 for iCODE version 3.0 or later. Enter a value from 0 to 1,999 for iCODE versions earlier than 3.0.

PCAM Master Point – If you selected Master_CAM_position, enter a value or expression for the PCam Master Point number, from:·0 to 12,999 for iCODE version 3.0 or later. Enter a value from 0 to 1,999 for iCODE versions earlier than 3.0.

= (value) – Enter the value or expression to assign to TCam Master Point.

Master CAM Time type

Select Master CAM type to assign a value to a Time Lock Cam Master Cam point. Make your entries in the following fields.

TCAM Master Point – Enter a value or expression for the TCam master point, from: 0 to 12,999 for iCODE version 3.0 or later. Enter a value from 0 to 1,999 for iCODE versions earlier than 3.0.

= (value) –Enter a value or expression to assign to the TCam Master point.

Indirect Variable

Select Indirect Variable to assign a value to an Indirect variable. Make your entries in the following fields.

Variable Address – Enter a value or expression for the variable's address (or number). For Indirect_Variable, the value can range from: 0 to 1,999 for iCODE version 3.0 or later. Values range from 0 to 999 for iCODE versions earlier than 3.0.

= (value) – Enter the value or expression to assign to the variable.

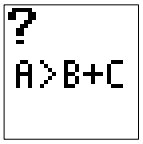
Inhibiting Resampling

If the value to be assigned is an expression, select Inhibit resample System Variables to prevent re-sampling of the system variables.

When you select inhibit resample system variables, system variables are not re-sampled when the expression is evaluated. If the value in the = field is not an expression, this selection has no effect.

When an expression is evaluated, all system variables are sampled simultaneously to provide a snapshot of the state of the controller at a specific instant in time. This ensures that consistent values are used in expressions that contain more than one system variable. In some cases, however, it is necessary to defeat this re-sampling and use the old (last sampled value) of a system variable in an equation. This is useful, for instance if an axis position must be used in multiple calculations—inhibiting the re-sampling ensures that the same position value is used for all calculations.

On Expression



Use the On Expression block to:

- Evaluate a mathematical expression and make a logical decision based on the expression's value (If Expression Type).
- Pause execution of a task or program until the value of a mathematical expression is greater than or less than zero (Wait For Expression Type).

The On Expression block resides on the Main Palette.

Wait for Expression

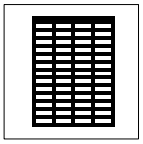
The Wait for Expression type pauses the program until the value of a mathematical expression meets the specified criteria. See the *Expression Builder* chapter in this manual for information on constructing expressions.

When you select the Wait For Expression type, the block has a single exit node.

If Expression

The If Expression type evaluates a mathematical expression and makes a logical decision based on its value. Program flow branches to the true node (at the top of the block) if the value of the expression does not equal zero, and to the false node (at the bottom of the block) if the expression equals zero. Enter the desired expression to evaluate.

Build Table



Use the Build Table block to build time and position cam profiles, and arrays of user variables. Construct a table of values by entering a value or expression for each item in the table, or by copying points from a spreadsheet.

See the *Online Help* for information on the Build Table Block Pop-Up Menu and for detailed instructions on how to take information from an existing spreadsheet and place it into the Build Table Block.

The Build Table block resides on the CAM Palette.

Building CAM Position Profiles

A CAM Position profile has two parts: a column of Master axis position values (in master axis position units) and a column of Slave axis position values (in slave axis position units). You can create these columns in the same Build Table block, or you can use a separate Build Table block for each column.

ATTENTION: If you use a separate Build Table block for each column, be sure to type the same value into the *Starting Offset* field in both blocks.

Each point in a CAM Position profile consists of two values, one for the master axis position (in the Master column) and one for the slave axis position (at the corresponding location in the slave column). GML Commander treats these point values as relative values. This means that, when the cam executes, GML Commander uses only the change between two points' profile values to generate motion. This lets you begin a cam starting at any master or slave axis position.

The cam tables in the motion controller can store any number of individual profiles of any length, provided the total number of points for all profiles does not exceed the capacity of the table. If you have iCODE version 3.0 or later, each cam table can contain up to 4000 points numbered 0 through 3999. If you have iCODE version 2.3 or earlier, each cam table can contain up to 2000 points numbered 0 through 1999.

Building CAM Time Profiles

A CAM Time profile has two parts: a column of Master Time values (in seconds) and a column of Slave axis position values (in slave axis position units). You can create these columns in the same Build Table block, or you can use a separate Build Table block for each column. If you use a separate Build Table block for each column, be sure to type the same value into the Starting Offset field in both blocks.

Each point in a CAM Position profile consists of two values, one for the master time value (in the Master column) and one for the slave axis position value (at the corresponding position in the slave column). GML Commander treats these point values as relative values. When the cam executes, GML Commander uses only the change between two points' profile values to generate motion. This lets you begin a cam starting at any master time or slave axis position.

The cam tables in the motion controller can store any number of individual profiles of any length, provided the total number of points for all profiles does not exceed the capacity of the table. For iCODE version 3.0 or later, each cam table can contain up to 4000 points numbered 0 through 3999. For iCODE version 2.3 or earlier, each cam table can contain up to 2000 points numbered 0 through 1999.

Building Variable Arrays

You can also use the Build Table block to assign values to a contiguous sequence of user variables for use with indirect addressing. In iCODE version 3.0 or later 2,000 user variables (from 0 to 1,999) are available. These can be used singly or as part of a table with indirect addressing. In iCODE version 2.3 or earlier, only 1,000 user variables (from 0 to 999) are available.

Starting Offset

Starting Offset relates to the list of position numbers, or points, in the Build Table block. These position numbers can run from:

- 0 to 3,999 for cam profiles using iCODE version 3.0 or later
- 0 to 1,999 for cam profiles using iCODE versions earlier than version 3.0

- 0 to 1,999 for variable arrays using iCODE version 3.0 or later
- 0 to 999 for variable arrays using iCODE versions earlier than version 3.0.

Starting offset does not relate to the axial position, time, or variable values stored in a Master or Slave table.

GML Commander stores the values associated with each position number, or point, at a defined address. This is the same way GML Commander stores user variable values. The address of each position number (or point), in a Build Table block's cam profile or variable array, is equal to the sum of the starting offset value plus the position number.

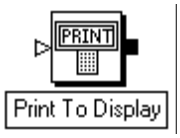
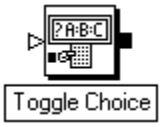
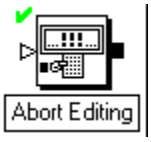
Use the starting offset value to avoid conflict between the addresses of previously defined user variables, and position numbers in a cam table or variable array. It is important that a position number (or point) address does not unintentionally conflict with a user variable address. If there is a conflict, the position number (or point) value overwrites the user variable value.

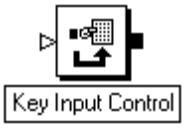
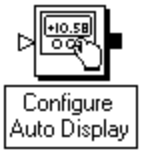
For example, if you have previously defined fifty user variables, with addresses at positions 0 through 49, your starting offset in the Build Table block would be a value not less than 50. But, if you failed to enter a starting offset value, the first 50 values in your new cam profile or variable array would also have addresses of 0 through 49, and would overwrite the previously defined user variables.

Display and Operator Interface Blocks

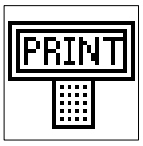
Display and Operator Interface blocks control the display and entering of information using an operator interface device connected to serial port B of the motion controller. Some of these blocks can also be used to interface to the GML Commander Terminal window on the host computer, using serial port A. When you select DH-485 Link in the General page of the Configure Control Options dialog box, serial port B on the motion controller is used for DH-485 communication and the normal built-in operator interface functions of GML Commander and the motion controllers are unavailable.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Display messages and variable values on the operator interface
	Create custom menus that let the operator change user variable values at runtime, in a manner similar to the motion controller's built-in operator interface setup menus.
	Create custom menus and prompt the operator to make a runtime selection from a scrolling list of choices, in a manner similar to the motion controller's built-in setup menus for an operator interface.
	Perform a controlled interrupt of a current task's ongoing editing session, and let another task take over control of the operator interface for a brief time.

Use this block:	To:
	Immediately update the display (General Watch or Tag Window) with the current values.
	• Control the serial port buffer pointer of serial port B to get the next or previous character in the buffer. • Purge any extraneous characters in the serial port buffer before sending a request for data to the serial port.
	• Pause the program until serial port B receives a character. • Check serial port B for the presence of a received character.
	Enable—and configure up to six fields that will appear in—the runtime display. The runtime display is an informational display that is constantly updated while the motion controller operates.

Print to Display



Use the Print to Display block to display messages and variable values on the operator interface. Select the operator interface in the Interface page of the Configure Control Options dialog box.

Each Print to Display block can present up to four fields on a single line, each field consisting of text, an ASCII control character, or an expression. This lets you display variable values with (for example) explanatory text. The fields are displayed in the sequence of their tabbed Field input pages.

The Print to Display block resides on the RS-232/422 Palette.

Unless you select Force to Port (see below), the message goes to the serial port currently configured for operator interface. See *Configuring Your Axis* in this manual, and the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on selecting the operator interface serial port.

Unless you select Suppress Auto CR/LF (see below), the message ends with a carriage return and a line feed character to properly scroll the display.

If you are not using an operator interface or host computer connected to one of the motion controller's serial ports, you do not need any Print to Display blocks in your GML Commander diagram.

Force to Port

To send text, values, or ASCII control codes to a specific serial port—regardless of which serial port is used for the operator interface—select Force to Port. Then select the desired serial port (A or B) from the pop-up menu. This causes the complete message generated by the Print to Display block to be sent to the specified serial port. If you selected DH-485 Link in the General page of the Configure Control Options dialog box, serial port B on the motion controller is used for DH-485 communication and cannot be used by the Print to Display block.

Suppress Auto CR/LF

If you select Suppress auto CR/LF, the trailing carriage return and line feed characters normally sent at the end of a message are not sent. Use this selection to display messages—generated by two different Print to Display blocks—on the same line. Because a single Print to Display block can generate a message with only four fields, you can use two Print to Display blocks (with Suppress Auto CR/LF selected in the first block) to display a single line on the display that is composed of more than four fields.

Selecting the Display

The Print to Display dialog box can consist of up to four field tabbed pages. Press the Add Another Field button to add an additional Field page, and the Remove This Field button to delete the current Field page.

The available display options are:

ASCII Text

To display standard ASCII text in any of the four fields, select ASCII Text in the Field menu, and enter the desired text in the data entry box for this field. ASCII text is always sent exactly as entered including any imbedded spaces, preserving upper and lower case characters. The ASCII text may be up to 120 characters long, but is usually limited to less than this by the width of the operator interface device.

Any legal ASCII character (see *Appendix B* for an ASCII reference table) except those shown in the following table may be used as part of an ASCII text field.

Illegal ASCII Text Field Characters

ASCII	Keyboard	Decimal	Hex	Description
LF	CTRL + J	10	0A	Line Feed
ESC	CTRL + [	27	1B	Escape
;	;	59	3B	Semi-colon
:	:	58	3A	Colon
“	“	34	22	Quote

To display any one of these characters, use the ASCII Code Field selection as explained in the following section.

ASCII Code

To send an ASCII control code (for cursor control, for example) or to display one of the illegal ASCII text characters in any of the four fields, select ASCII Code from the Field menu. Then enter the ASCII code, for the desired character, in the data entry box. The ASCII code entry is a decimal number between 0 and 127 inclusive. See *Appendix B* for an ASCII reference table showing the ASCII codes for all characters.

If you selected ASCII Code for the last displayed field in the Print to Display block, the trailing carriage return and line feed characters are not sent, regardless of the Suppress Auto CR/LF selection.

Expression

To display the current value of any variable or expression as part of the message, select Expression from the Field menu, and enter the desired expression in the data entry box for this field.

Values are displayed using a format determined by the motion controller, based on the number of significant digits in the value. To directly specify the format (and thus limit the number of characters used), select Formatted Expression for the field as explained below.

Formatted Expression

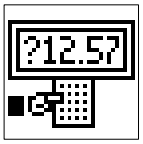
To display the current value of any variable or expression using a specific numeric format as part of the message, select Formatted Expression from the Field menu. Then enter an expression in the data entry box for this field, and a numeric format in the Format data entry box.

Enter the desired format for the value using the # character as a placeholder to denote significant digits. Use a decimal point (.), if decimal information is to be displayed. To display the value as a positive or negative signed quantity, place a + character at the beginning of the numeric format. Without the + character, only the magnitude of the value is displayed. For example, a format of ### allows positive integer values between 0 and 999 inclusive to be displayed, while a format of +### allows integer values between –999 and +999 to be shown.

The number of characters displayed in a formatted expression field is always the same and equal to the total number of characters in the format regardless of the value. Leading spaces and trailing zeros are automatically added to fill out the defined format.

If the absolute value of the variable or expression is too large to be displayed using the specified format, it is displayed as a field of Xs. For example, if a Format of +###.### is used to display a variable or expression with a current value 1000.3, the resulting display is +XXX.XXX.

Edit Value



Use the Edit Value block to create custom menus that let the operator change user variable values, in a manner similar to the motion controller's built-in operator interface setup menus. These custom menus are displayed on the operator interface port selected in the Interface page of the Configure Control Options dialog box.

The Edit Value block resides on the RS-232/422 Palette.

When the Edit Value block executes, it prompts the operator for a new user variable value. A prompt message appears on the operator interface (either the GML Commander Terminal Window via Port A, or a separate operator interface terminal via Port B), followed by the current value of the selected user variable. The operator then enters a new value and presses ENTER, which updates the value of the specified user variable. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the operator interface.

If other tasks are executing (multitasking), the task that contains this block pauses while the operator enters the new variable value, but the other tasks continue to execute. In this way, an Edit Value block in one task does not pause execution of any other task, or hang the task dispatcher.

While the operator is entering the new value, mistakes can be corrected using CLEAR on the integrated Setup and Diagnostic Panel on the IMC-S/21x controllers, or the Delete or Backspace keys on a terminal.

Prompts

The prompt message may be up to 24 characters long and may contain any legal ASCII character (see Appendix A for an ASCII reference table) except those shown in the following table.

Illegal ASCII Text Field Characters

ASCII	Keyboard	Decimal	Hex	Description
LF	CTRL + J	10	0A	Line Feed
ESC	CTRL + [	27	1B	Escape
;	;	59	3B	Semi-colon
:	:	58	3A	Colon
“	“	34	22	Quote

Value Display Format

Enter the Value Display Format using the # character as a place holder to denote significant digits. Use a decimal point (.) as necessary for decimal information. To display the value as a positive or negative signed quantity, place a + character at the beginning of the value display format. Without the + character, only the magnitude of the value is displayed. For example, a format of ### displays integers between 0 and 999 inclusive, while a format of +### displays integer values between –999 and +999.

When you use a value display format, the number of characters used to display a value is always the same, regardless of the value. Leading spaces and trailing zeros are automatically added to fill out the defined format. If the current absolute value of the variable is too large to be displayed using the specified value display format, it is displayed as a field of Xs. For example, if a value display format of ###.### is used to display a variable with a current value 1000.3, the resulting display is +XXX.XXX. Note that the value display format affects only the displayed value, not the entered value. See *Range Limiting* for more information on limiting the range of entered values.

Range Limiting

To limit the range of values that the operator can enter for the user variable, select Range Limit Input and enter the desired values for the smallest and largest allowed values in the appropriate data entry boxes. If the operator enters a value outside these limits, the value is prompted for again.

Tag Explorer

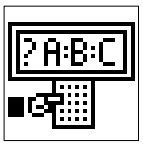
User Variables is pre-selected, by default.

Tag Window

Select the user variable whose value is to be edited during runtime.

Note: If a user variable does not appear, right click in the List window and select **New** to create a new user variable.

Toggle Choice



Use the Toggle Choice block to create custom menus and prompt the operator to make a runtime selection from a scrolling list of choices, in a manner similar to the motion controller's built-in setup menus for an operator interface. The toggle choice menus are displayed on the operator interface port selected in the Interface page of the Configure Control Options dialog box.

The Toggle Choice block resides on the RS-232/422 Palette.

When the Toggle Choice block executes, a prompt message appears on the operator interface (either the GML Commander Terminal Window via port A, or a separate operator interface terminal via port B), followed by the currently selected choice. To scroll through the available choices, the operator should press NEXT or ± on the integrated Setup and Diagnostic Panel on the IMC-S/21x controllers, or the Space bar on a terminal. The choices are displayed sequentially in the order shown in the scrolling choices list. When the desired selection appears, pressing ENTER selects it, which updates the value of the selected variable. Decisions can then be made based on the value of the selection variable in subsequent blocks.

See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the operator interface.

If other tasks are executing (multitasking), the task that contains this block pauses while the operator chooses the item, but the other tasks continue to execute. In this way, a Toggle Choice block in one task does not pause execution of any other tasks or hang the task dispatcher.

Prompts

The prompt message may be up to 24 characters long and may contain any legal ASCII character (see *Appendix A* for an ASCII reference table) except those shown in the following table.

Illegal ASCII Text Field Characters

ASCII	Keyboard	Decimal	Hex	Description
LF	CTRL + J	10	0A	Line Feed
ESC	CTRL + [	27	1B	Escape
;	;	59	3B	Semi-colon
:	:	58	3A	Colon
“	“	34	22	Quote

Entering Choices

To define the choices, enter the desired text for the first choice in the data entry box to the right of the number 0. Each choice can be of any length and can include embedded spaces. Up to 10 choices may be specified for use with a single Toggle Choice block.

For example:

To define the:	Type:
First choice as no	No in the first field.
Second choice as yes	Yes in the field to the right of the number 1.

You can enter up to 10 choices (selection variable values of 0 – 9) in this manner.

At runtime, when the operator selects No, the value 0 is assigned to the selected user variable; when the operator selects Yes the value 1 is assigned to the selected user variable.

Editing the Choices

The current choices are shown in the scrolling choices list. The list can be edited by selecting the desired choice, and entering a new message in the data entry box.

Tag Explorer

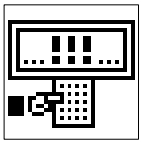
User Variables is pre-selected, by default.

Tag Window

Select the user variable that stores the value corresponding to the choice selected from the toggle menu during runtime.

Note: If a user variable does not appear, right click in the List window and select **New** to create a new user variable.

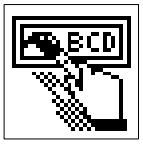
Abort Editing



Use the Abort Editing block during multitasking to perform a controlled interrupt of a current task's ongoing editing session, and let another task take over control of the operator interface for a brief time (to handle faults, for example). The Abort Editing block performs this interruption in a controlled manner. When the interrupting task finishes using the operator interface, the Abort Editing block cleans up the display, allowing the interrupted editing session to resume.

The Abort Editing block resides on the RS-232/422 Palette. It requires no parameters, and has no dialog box. This block requires no parameters and it always has a check mark by the upper left corner of the block to indicate that it is complete.

Refresh Display

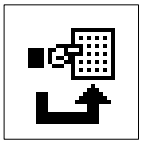


Use the Refresh Display block to immediately update the runtime or status display with the current values of the displayed variables, axis status, etc. It is most useful in forcing the display to show an immediate update to a particular variable rather than waiting for the configured display refresh time.

See the *Setup* section in the *Installation and Setup* manual for your motion controller for more information on configuring the runtime display.

The Refresh Display block resides on the RS-232/422 Palette. This block requires no parameters, has no dialog box, and it always has a check mark by the upper left corner of the block to indicate that it is complete.

Key Input Control



Use the Key Input block to:

- Control the serial port buffer pointer of serial port B to get the next or previous character in the buffer.
- Purge any extraneous characters in the serial port buffer before sending a request for data to the serial port.

When you select DH-485 in the General page of the Configure Control Options dialog box, the Key Input Control block cannot be used, because serial port B on the motion controller is used for DH-485 communication.

The Key Input block resides on the RS-232/422 Palette.

The motion controller maintains a 120-character buffer for serial port B. As characters are received, the motion controller stores them in the next sequential location in the buffer. The Get Next Key and Get Previous Key functions of the Key Input Control block are used to step through the received characters to find a specific character, when multiple characters have been received. This—in conjunction with the Last_Keypress system variable, which stores the actual character value—makes it possible to implement a complete, custom operator interface menu in a GML Commander diagram.

If the serial port buffer fills up before the all the characters have been processed (or the buffer cleared), serial port B transmits an XOFF character (CTRL-S or ASCII 19) to stop the sending device from sending more characters. When the serial buffer has been emptied sufficiently to allow more characters to be received, an XON (CTRL-Q or ASCII 17) is transmitted. This signals the transmitting device that it may resume sending characters to the motion controller.

Clearing the Input Buffer

Selecting Clear Input Buffer from the Type menu immediately clears the input buffer of serial port B, discarding any characters therein. You can use this function at any time to clear the serial port buffer, but that you should first clear the serial buffer before requesting any data from the operator. This guarantees that the only those characters you want—and not any extraneous ones—are evaluated by subsequent If Key blocks.

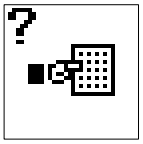
Getting the Next Key

Selecting Get Next Key from the Type menu loads the ASCII value of the next key in the buffer into the Last_Keypress system variable. The key (or character) can then be evaluated and a decision made using an If Key block. If no character has been received yet, or there are no more characters in the buffer, Last_Keypress is zero.

Getting the Previous Key

Selecting Get Previous Key from the Type menu loads the ASCII value of the previous key into the Last_Keypress system variable. The key (or character) can then be evaluated and a decision made using an If Key block. While it is not usually necessary to go backwards in the serial port input buffer using this function, it can be useful in some situations.

On Key Press



Use the On Key Press block to:

- Pause the program until serial port B receives a character (Wait for Key type).
- Check serial port B for the presence of a received character (If Key type).

If you selected DH-485 in the General page of the Configure Control Options dialog box, the On Key Press block cannot be used, because serial port B on the motion controller is used for DH-485 communication.

The On Key Press block resides on the RS-232/422 Palette.

Wait for Key

The Wait for Key type pauses the program until serial port B receives a character. It can wait for any key, a specific key, or a specific ASCII code. With Wait for Key type selected, the On Key Press block has a single exit node.

If other tasks are executing (multitasking), the task containing this block pauses, while the other tasks continue to execute. In this way, a Wait for Key type block in one task does not pause execution of any other task, or hang the task dispatcher.

If Key

The If Key type checks serial port B for the presence of a received character. It can check for any key, a specific key, or a specific ASCII code. With If Key type selected, the On Key Press block has two exit nodes. Program flow branches to the true node (at the top of the block) if a key has been received, and to the false node (at the bottom of the block) if not. Select the key type to check for from the pop-up menu.

Waiting or Checking for the Next Key

Use the On Key Press block to wait for or to check the next key in the input buffer. To do this, be sure to select the *Next Key* field.

If you are using the On Key Press block to step through the input buffer, be sure not to select Next Key. Not selecting Next Key guarantees that the input buffer location, selected by the last Key Input Control block—not the next key in the buffer, is checked (If Key type) or used (Wait for Key type).

Waiting or Checking for Any Key

The effect of selecting Any Key from the Key Type menu depends upon your Type selection (Wait for Key or If Key), as follows.

Checking for Any Key

Selecting Any Key, with If Key selected, causes program flow to branch to the true node (at the top of the block) if any keystroke character has been received by serial port B. If no character has been received, program flow branches to the false node (at the bottom of the block). This selection is useful for determining that a key has been pressed or a character received before evaluating it.

Waiting for Any Key

Selecting Any Key, with Wait for Key selected, pauses the program until any keystroke character has been received by the serial port B. This selection is useful for waiting for a keystroke character to be received before evaluating it.

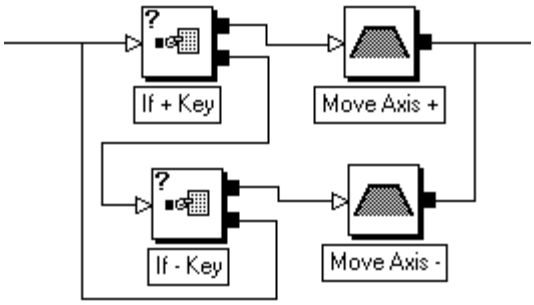
Waiting or Checking for a Specific Key

The effect of selecting Specific Key from the Key Type menu depends upon your Type selection (Wait for Key or If Key), as follows.

Checking for a Specific Key

Selecting Specific Key, with If Key selected, causes program flow to branch to the true node (at the top of the block) only if the specific character, entered in the data entry box, has been received by serial port B. If any other character (or no character) has been received, program flow branches to the false node (at the bottom of the block).

This selection is useful for interpreting a received keystroke character before taking specific action. For example, the following example checks for either a + or – character to be received and then moves an axis in the appropriate direction.



Waiting for a Specific Key

Selecting Specific Key, with Wait for Key selected, pauses the program until the specific character entered in the Key data entry box has been received by serial port B.

Checking or Waiting for an ASCII Code

The effect of selecting ASCII Key from the Key Type menu depends upon your Type selection (Wait for Key or If Key), as described below. Enter the ASCII code for the desired character in the data entry box as a decimal number between 0 and 127 inclusive. See *Appendix B* for an ASCII reference table showing the ASCII codes for all characters.

Checking for ASCII Code

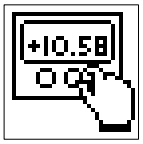
Selecting ASCII Key, with If Key selected, causes program flow to branch to the true node (at the top of the block) only if the specific ASCII character code—entered in the data entry box—has been received by serial port B. If any other character (or no character) has been received, program flow branches to the false node (at the bottom of the block).

Waiting for ASCII Code

Selecting Wait for ASCII Code, with If Key selected, pauses the program until the specific ASCII character code—entered in the data entry box—has been received by serial port B.

This selection is useful to wait for non-printing characters or ASCII control codes.

Configure Auto Display



Use the Configure Auto Display block to enable and configure up to six fields that appear in the runtime display. The runtime display is an informational display that is constantly updated while the motion controller operates.

The selected fields appear in the runtime display from left to right on a single line, in the order in which they are defined in the Configure Auto Display dialog box.

The runtime display can appear either in GML Commander's Terminal Window (via serial port A), or independently in a separate operator interface (via serial port B). Make this selection in the Interface page of the Configure Control Options dialog box, along with the setting for the runtime display refresh rate.

When DH-485 is selected in the General page of the Configure Control Options dialog box, serial port B on the motion controller is used for DH-485 communication making the normal built-in operator interface functions of GML Commander and the motion controllers unavailable.

The Configure Auto Display block resides on the RS-232/422 Palette.

Configuring Runtime Display Fields

You may elect to display any one of five different display types in each one of the six available runtime display fields:

- Axis Status
- Numeric Value (variables and expressions)
- Global Fault
- System Fault (1394 motion controllers, only)
- Axis Fault

Field Legends

Each field in the runtime display has an associated label, or legend. The legend may be up to 24 characters long and may contain any legal ASCII character (see *Appendix B* for an ASCII reference table) except those shown in the following table

Illegal ASCII Legend Characters

ASCII	Keyboard	Decimal	Hex	Description
LF	CTRL-j	10	0A	Line Feed
ESC	CTRL-[	27	1B	Escape

Axis status is displayed as a text message as shown in the following table. The status conditions are prioritized from highest to lowest in the order shown in the table. The higher the status value, the higher its priority (or severity). In other words, a hardware overtravel fault has higher priority than a software travel limits fault.

Axis Status Field Messages

Runtime Display	Description
AXL FLT	AxisLink Timeout
AXL FLT	AxisLink Fault (axis not found)
AXS OFF	Virtual Axis Not Enabled
DRV FLT	Drive Fault
ERR FLT	Position Error Tolerance Fault
HRD LIM	Hardware Overtravel Fault
SFT LIM	Software Travel Limits Fault
ENC FLT	Encoder Noise or Loss Fault
SRV OFF	Feedback OFF or Virtual Axis Enabled
OUT LIM	Servo Output Limited

Axis Status Field Messages

Runtime Display	Description
HOMING	Homing
MOVING	Moving or Executing Time-Lock Cam
JOGGING	Jogging
UNLOCK	Axis Unlocked
LOCKED	Axis Locked

When a given axis status is displayed, any status condition of lower priority can also be active. For instance, if axis status is shown as HRD LIM, a SFT LIM condition may also be active and feedback may be OFF. When the highest priority fault is cleared, the next highest priority active status value appears in the field.

When entering an input format, use the # character as a place holder to denote significant digits. Use a decimal point (.) to display decimal information. To display the value as a positive or negative signed quantity, place a + character at the beginning of the numeric format. Without the + character, only the magnitude of the value is displayed. For example, a format of ### displays integers between 0 and 999, while a format of +### displays integer values between –999 and +999.

The number of characters displayed in a numeric value field is always equal to the total number of characters in the format data entry box regardless of the value. Leading spaces and trailing zeros are automatically added to fill out the defined format.

If the absolute value of the variable or expression is too large to be displayed using the specified format, it is displayed as a field of Xs. For example, if a format of +###.### is used to display a variable or expression with a current value 1000.3, the resulting display is +XXX.XXX.

All faults—global, system or axis—are displayed as a text message as shown in the following tables.

Motion Controller Faults

Runtime Display	Description	Global	Axis	1394 Axis	1394 System
AXES OK	No Faults	✔			
AXIS OK	No Faults		✔	✔	
AXL FLT	AxisLink Virtual Axis Link Lost	✔	✔		
AXL FLT	AxisLink Virtual Axis Fault	✔	✔		
BUS FLT	Bus Loss Fault			✔	✔
BOV FLT	Bus Over Voltage Fault				✔
CTR FLT	Control Power Loss				✔
DH FLT	DH-485 Fault	✔			
DRV FLT	Drive Fault	✔	✔		
ENC FLT	Encoder Fault		✔		
ENC FLT	Encoder Noise or Loss Fault	✔			
ENC FLT	Encoder/Resolver Fault			✔	
ERR FLT	Servo Error Fault	✔	✔		
ERR FLT	Position Error Fault			✔	
GND FLT	Ground Fault				✔
HRD FLT	Hard Travel Limit Fault	✔	✔	✔	
I-T FLT	It Fault			✔	
MEM FLT	Setup Data Memory Fault	✔			
MTR FLT	Motor Thermal Fault			✔	
NO AXES	Not Installed			✔	

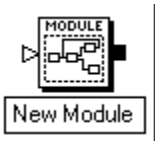
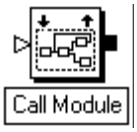
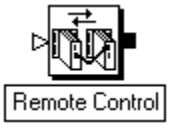
Motion Controller Faults

Runtime Display	Description	Global	Axis	1394 Axis	1394 System
PHS FLT	Phase Loss Fault				✔
PRG FLT	Application Program Memory Fault	✔			
PWR FLT	Power Fault			✔	
RIO FLT	RIO Fault	✔			
RNG FLT	Ring Fault			✔	✔
SFT LIM	Software Travel Limits Exceeded	✔	✔	✔	
SYST OK	No Faults				✔
TMP FLT	Over Temperature Fault				✔

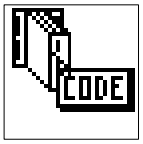
Miscellaneous Blocks

Miscellaneous blocks, given their varying functionality, do not fit into the categories previously discussed. These blocks are not frequently used.

The following table presents the icons for each block in this category along with a brief description. More detailed descriptions of the blocks are contained on the pages following this table.

Use this block:	To:
	Directly enter iCODE (the native language of the motion controllers) commands into your GML Commander diagram.
	Create a new module that can become part of your GML Commander diagram, or serve as an external module that can be called by your GML Commander diagram using a Call Module block.
	Call and execute an external module as part of your GML Commander diagram.
	Let a master motion controller, using Multidrop, request a user variable value, status condition, or fault condition from another motion controller.

Native Code

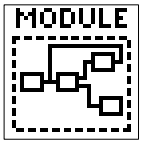


You can use the Native Code block to directly enter iCODE (the native language of the motion controllers) commands into your GML Commander diagram. However, few GML Commander diagrams ever require this block.

Only experienced programmers should use the Native Code block. Its purpose is to let you enter iCODE commands supported by your firmware, but not by GML Commander. Contact Allen-Bradley Technical Support if you believe you need to use the Native Code block to accomplish something that cannot be done any other way.

The Native Code block resides on the Advanced Motion Palette.

New Module



Use the New Module block to create a new module that can become part of your GML Commander diagram, or serve as an external module that can be called by your GML Commander diagram using a Call Module block.

By itself, a New Module block provides no program functionality. It is merely a container that you can use to hold other function blocks.

You can create a New Module block in either of two ways:

- Drag a New Module block from the Main Palette, and drop it into the diagram window, or
- Encapsulate other function blocks, by holding down the left mouse button, dragging a dotted rectangle around other function blocks (that are already in your diagram), and selecting Encapsulate from the Module menu.

When you add a New Module to a diagram, that New Module instantaneously appears in the Diagram Explorer as part of the current diagram.

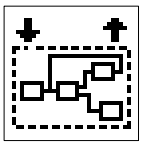
Open the New Module block either by double-clicking on it, or by clicking on the module name in the Diagram Explorer.

To close the module and return to the New Module block, do one of the following:

- In the Diagram Window, click the right mouse button and select Collapse Module
- In the Diagram Explorer, left click on the diagram or module that contains the expanded module

The New Module block resides on the Main Palette. It requires no parameters, and has no dialog box. When you successfully translate the diagram to iCODE script (using the Translate to Script command in the Diagram menu) a check mark appears at the upper left corner of the module block.

Call Module



Use the Call Module block to call and execute an external module as part of your GML Commander diagram. The Call Module block evaluates a value or expression that you type into the Call Module dialog box, and calls for the module associated with the resultant value (e.g., Module numbers 0, 1, 2, etc.).

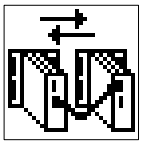
The Call Module Block evaluates a value or mathematical expression and executes the external module associated with that value as a subroutine in your diagram.

The Call Module block can call only a module that you have previously added to the Call Module list. If the value of the expression you type into the Call Module block evaluates to a value not listed in the Call Module Tag window, a runtime error occurs.

To avoid very complex diagrams, do not use the Call Module Block unless the application absolutely requires it. The use of this feature may make it very difficult to edit and debug program flow.

You may wish to use the Call Module block when an external device needs to make program flow decisions. The external device requires a method of labeling the routines that the motion controller must execute. The Call Modules selection in the Tag Explorer displays, in the Tag Window, the list of available modules that can be used to program the external device. The external device can then send a variable containing the label of the routine it needs to execute. The Call Module block can evaluate this variable and call the appropriate predefined subroutine module as part of the application program.

Remote Control



Use the Remote Control block to let a master motion controller, using Multidrop, request a user variable value, status condition, or fault condition from another motion controller.

The Remote Control block resides on the RS-232/422 Palette.

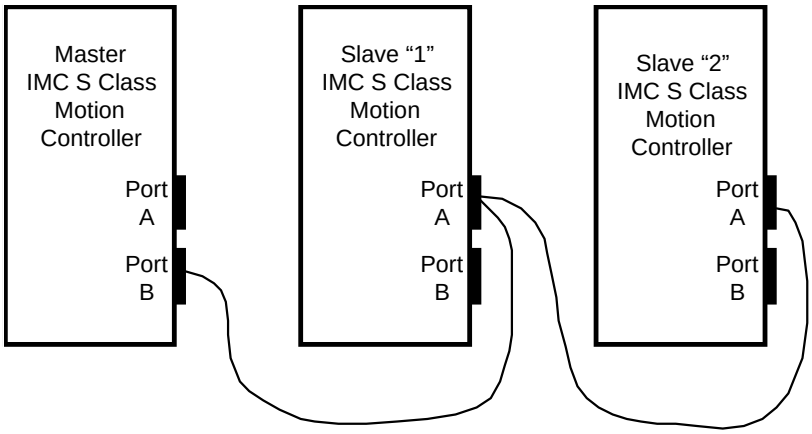
Unless you select Force to Port (see *Specifying the Serial Port*), the Remote Control block sends a specified variable or expression to query, from serial port B of the master motion controller to serial port A of the selected slave motion controller. The slave motion controller evaluates the expression and sends the result back to the master motion controller. The master motion controller then stores the received value in a specified variable, using an optional response format.

If you selected DH-485 in the General page of the Configure Control Options dialog box, the master motion controller uses serial port B for DH-485 communication, and must use Port A for multidrop communications. In this case, specify Force to Port A for remote control operation.

Multidrop

Multidrop is a communications scheme that allows multiple motion controllers to communicate with each other over a single RS-422 serial link. When using multidrop, serial port B of the master motion controller is connected to serial port A of the slave controllers, as shown below.

If DH-485 is selected do not use this scheme of master slave multidrop. Instead, you must daisy chain together serial ports A.



The GML Commander diagram in the master motion controller uses Remote Control blocks to read variable values and/or status conditions from a selected slave controller.

For proper operation, each slave motion controller must have multidrop mode enabled, either in the General page of its Configure Control Option dialog box or via an Adjust type command within a Control Setting block. In addition, each slave motion controller must have a valid address selected using the rotary ADDRESS switch on the front panel. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on enabling multidrop mode.

Remote Controller Address

Select the address of the desired slave motion controller from the menu. Addresses are set using the rotary Address switch on the front panel of the motion controller. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring slave motion controllers for multidrop operation.

Expression to Query

Enter the desired slave user variable, status variable, fault variable, or expression in the *Expression to Query* field. The user variable definitions in the slave motion controller are not known to the master, therefore use the Indirect_variable function to query a user variable.

GML Commander sends the entered expression to a slave motion controller for evaluation, so do not include in the Expression to Query any user variables, constants, or I/O. These items were defined for use only within the master motion controller GML Commander diagram, and may be meaningless to the slave motion controller.

Although you request a single variable value most frequently, you can also use an expression composed entirely of constants, slave motion controller user variables, status variables, and fault variables.

Response Format

Type the desired format for the slave motion controller's response in the *Response Format* field using the # character as a placeholder to denote significant digits. Use a decimal point (.) as required if fractional information is expected. When querying positive or negative signed variables, place a + character at the beginning of the response format. Without the + character, only the magnitude of the value is returned.

If the current value of the requested expression cannot be represented using the specified response format, the slave motion controller returns a field of Xs. For example, if a Response Format of ###.### is used to request a variable with a current value 1000.3, the resulting message from the slave is XXX.XXX. This generates an Illegal Numeric Format runtime fault (Runtime_fault = 10) in the master motion controller, and the program is aborted. The GML Commander diagram for the slave controller must ensure that all possible values, for the expression requested by the master controller using the Remote Control block, are representable by the specified response format.

Specifying the Serial Port

To send the Expression to Query from a specific serial port, select Force to Port and select the desired serial port (A or B). This causes the complete query generated by the Query Remote Control block to be sent to the selected serial port.

Destination Page

The Destination Page of the Remote Control Block is for setting the user variables in the Tag Explorer and the Tag Window.

Tag Explorer

User Variables is pre-selected, by default.

Tag Window

Select the user variable that stores the value corresponding to the choice selected from the toggle menu during runtime.

Note: If a user variable does not appear, right click in the List window and select **New** to create a new user variable.

Chapter 23

Defining Variables, Constants, and I/O

The Tag Explorer and the Tag Window provide areas in which to view and select axes system variables and general system variables, and to define and select user constants, user variables and I/O.

This chapter contains step-by-step procedures for viewing your program's variables, constants, and I/O values. The topics in this chapter are:

- Displaying system variables.
- Defining a user variable/constant.
- Defining flex inputs and outputs.
- Defining inputs and output for RIO, SLC, AxisLink, and DH-485
- Using the general watch function.

Understanding System Variables

A variable is a defined place in the controller's memory that is allocated to store a number. System variables have predefined meanings. System variable values are determined during initial program setup, and can change when you program and execute your diagram.

There are two types of system variables:

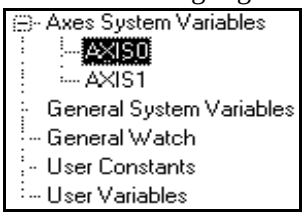
Type:	Description:
Axis	Display information related to a defined axis.
General	Display non-axis-related information, for example, data relating to the CPU, I/O communication program, fault, task, and fault conditions.

You can view the values assigned to system variables in the Tag Window. You can also view the variables using the General Watch function as they dynamically respond to the execution of the program. (See *Using the General Watch Function* in this chapter.) The Expression Builder also offers numerous system variables and system functions based on your controller configurations.

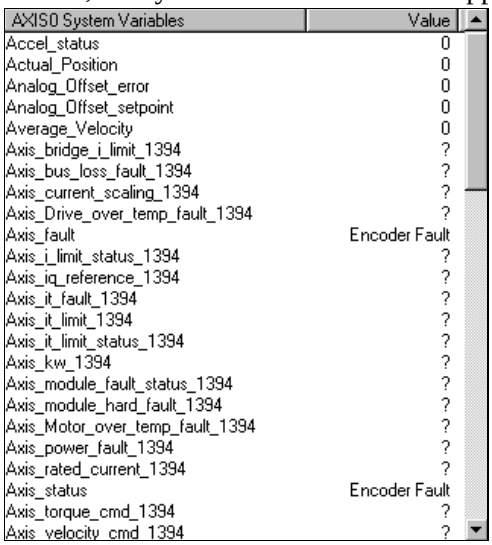
Example of How to Display a System Variable

In the following example, an axis-related system variable is selected for display. Use the same method to display a nonaxis-related (general) system variable.

1. In the Tag Explorer, select **AXIS0**. The system variable category AXIS0 is highlighted:



In the Tag Window, the system variables of axis 0 appear:



AXIS0 System Variables	Value
Accel_status	0
Actual_Position	0
Analog_Offset_error	0
Analog_Offset_setpoint	0
Average_Velocity	0
Axis_bridge_i_limit_1394	?
Axis_bus_loss_fault_1394	?
Axis_current_scaling_1394	?
Axis_Drive_over_temp_fault_1394	?
Axis_fault	Encoder Fault
Axis_i_limit_status_1394	?
Axis_iq_reference_1394	?
Axis_it_fault_1394	?
Axis_it_limit_1394	?
Axis_it_limit_status_1394	?
Axis_kw_1394	?
Axis_module_fault_status_1394	?
Axis_module_hard_fault_1394	?
Axis_Motor_over_temp_fault_1394	?
Axis_power_fault_1394	?
Axis_rated_current_1394	?
Axis_status	Encoder Fault
Axis_torque_cmd_1394	?
Axis_velocity_cmd_1394	?

2.

To:	Do the following:
View other variables	Scroll down the Tag Window to view another system variable.
Select a variable to add to the General Watch window	1. Select a system variable. 2. In the Tag Window, select a variable with the right mouse button. The variable short-cut menu appears. 3. Select **Watch** to add the selected variable to the General Watch Window.

Defining a User Variable/Constant

A variable or constant is a memory location allocated to store a number. GML Commander allows you to use user variables and constants in your program. You can set or modify these values during run-time or test and/or set variables in your program. You can test variables for a variety of conditions, for example, greater than (>), less than (<), or equal (=) conditions.

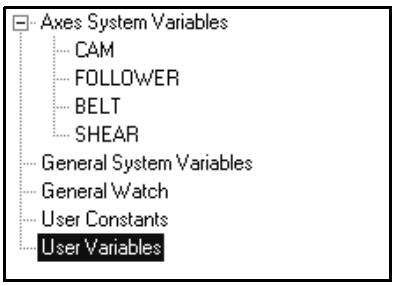
You must define user variables or constants before they appear as selections in the Tag Explorer. You define them for you application as you need them.

Creating, Editing, and Deleting a User Variable or Constant

With the following example, we will show you how to create a user variable, edit or delete the value, and add it to the General Watch Window. You can use the same methods for a user constant.

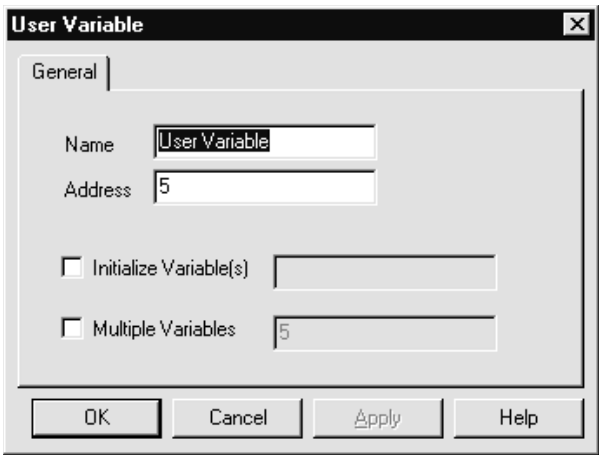
Note: You can define user variables to appear in the tag window even if the diagram has not been downloaded.

1. In the Tag Explorer, select **User Variable**. The User Variables category is highlighted in the Tag Explorer.



2. In the Tag Window, click your right mouse button. A variable short-cut menu appears.

3. Select **New**. The User Variable dialog box appears.



Note: Use the same variable short-cut menu to open the variable for editing. Double clicking in the Tag window opens a new tag, or double clicking on a predefined variable opens it for editing.

4. Make entries in the following fields:

Field	Description
Name	Type a unique variable name.
Address	Type the variable address.
Initialize variable(s)	If you select this variable, type a value for the user variable to be set to when the diagram is only downloaded to the controller.
Multiple variables	If you select this variable, type the number of variables that share this variable's name.

5. With the user variables displayed in the Tag Window:

To:	**Do the following:**
View changes in a user variable	In the Tag Window, scroll down to view changes to the user variable value as the program executes.
Add a user variable to the General Watch window	1. In the Tag Window, select a user variable with the right mouse button. The variable short-cut menu appears. 2. Select one of the four **Watches**. The variable is added to that Watch Window.
Delete a user variable from the Tag Window	1. In the Tag Window, select a user variable with the right mouse button. The variable short-cut menu appears. 2. Select **Delete**. The variable is deleted from the Tag Window listing.
Edit a user variable	1. In the Tag Window, select a user variable with the right mouse button. The variable short-cut menu appears. (You can also use the left mouse button to double click on the user variable. The User Variable dialog box appears.) 2. Select **Edit**. The User Variable dialog box appears. 3. Select parameters or type values to edit the variable. 4. Select **OK**.

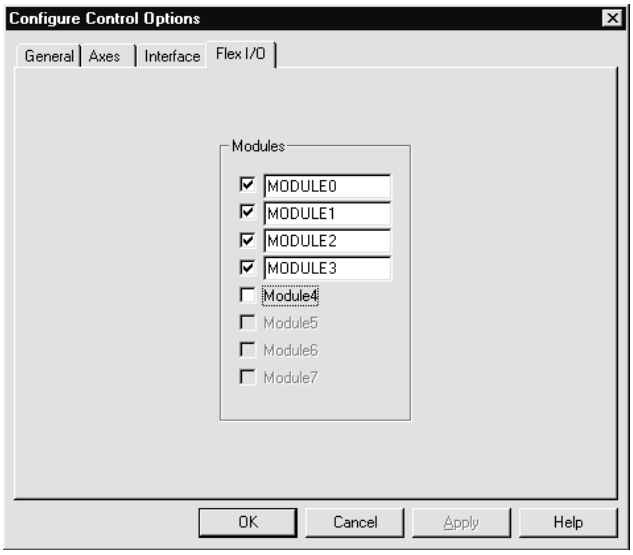
Note: You can sort user variables by name or address by using the left mouse button to click on the appropriate column header in the Tag Window.

Defining Flex Inputs and Outputs

You can connect up to eight separate Flex I/O modules to the motion controller in any order.

To define the inputs and outputs for Flex I/O modules:

1. Select **Control Options** from the Configure menu. The Configure Control Options dialog box appears
2. Select the **Flex I/O** tab to access the Flex I/O page.



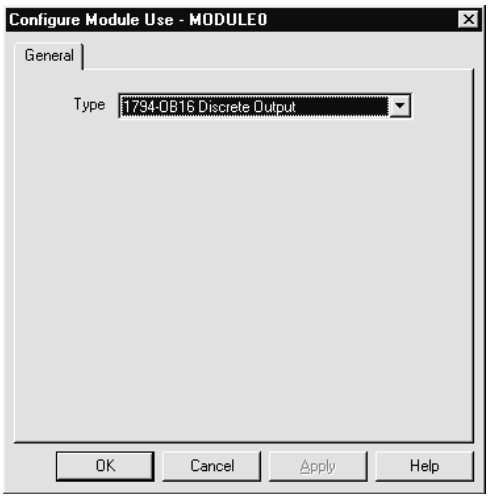
3. In the *Modules* area, make an entry in the following field:

Field	Description
Module*x*	1. Select the module that you want to use. 2. Type a new name for the module.

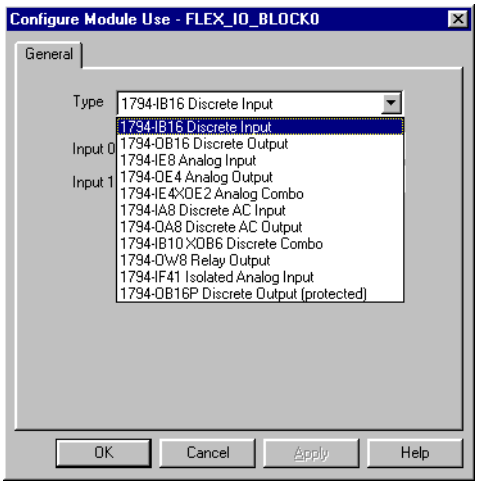
4. Select **OK**. The Configure Control Options dialog box closes.

5. Select **Flex I/O** from the Configure menu to access the Modules menu.

6. Select **MODULE0**. (MODULE0 is the default name. If you type a new name for this module, then this name appears.) A dialog box similar to the following appears:

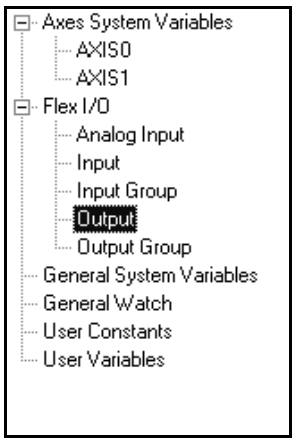


7. In the *Type* field, select an input, output, or combo value



Note: Some value selections require you to complete a second dialog box for defining parameters.

8. Select **OK**. The type of input, output, or combo value appears as a sub-category in the Tag Explorer under Flex I/O.

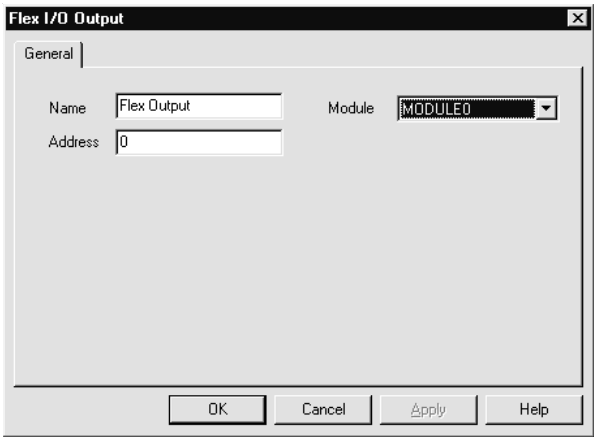


Creating Flex I/O Values

To create Flex I/O values:

1. In the Tag Explorer, select any Flex I/O category. The category name is highlighted in the Tag Explorer and the category name appears in the title bar at the top of the Tag Window.
2. Select the Tag Window with the right mouse button. The variable short-cut menu appears:

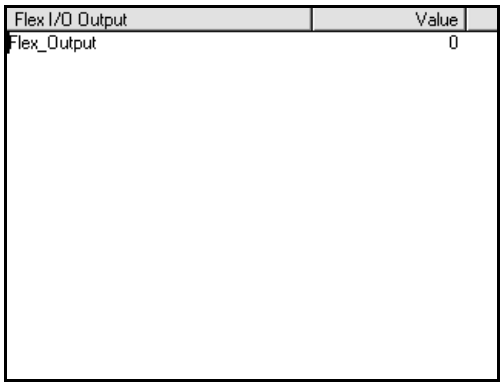
3. Select **New**. The Flex I/O dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	The name of the Flex I/O input or output that you are creating.
Address	The location of the Flex I/O input or output.
Module	The name of the module.

5. Select **OK**. The value appears in the Tag Window.



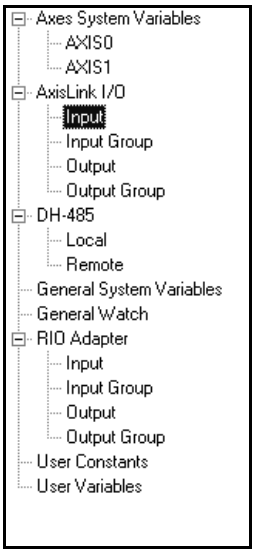
Defining I/O for RIO, SLC, AxisLink, or DH-485

Use the same method for the optional RIO, SLC, AxisLink, or DH-485 as you did to add input and output categories that appear in the Tag Explorer. After selecting the check box for an optional interface on the Configure Control Options dialog box, use the following section to define each input or output.

Note: We are using AxisLink to illustrate the steps to define the inputs and outputs that appear in the Tag Explorer and the Tag Window.

To create input and output values:

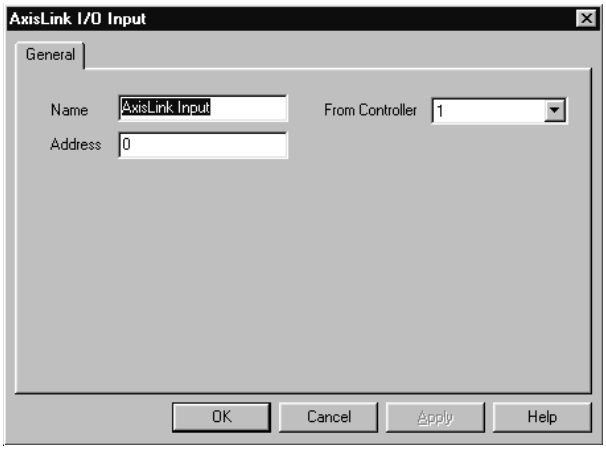
1. In the Tag Explorer, select any input or output category. The category name is highlighted in the Tag Explorer:



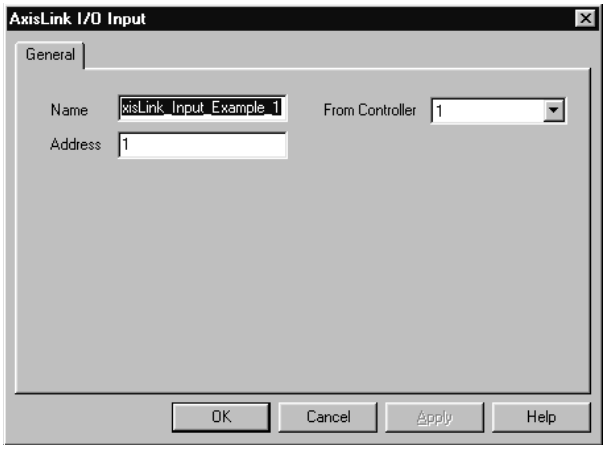
The category name appears in the title bar at the top of the Tag Window:

2. Select the Tag Window with the right mouse button. The variable short-cut menu appears

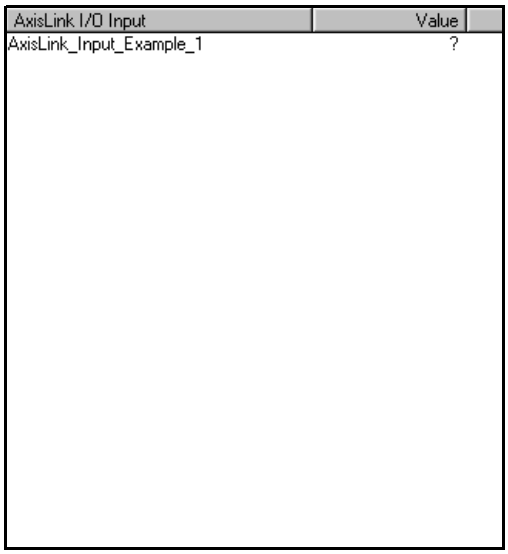
3. Select **New**. The dialog box for the selected input or output appears:



4. Make the appropriate entries in fields for the selected interface. The name of the AxisLink I/O Input that is entered is shown in the example below:

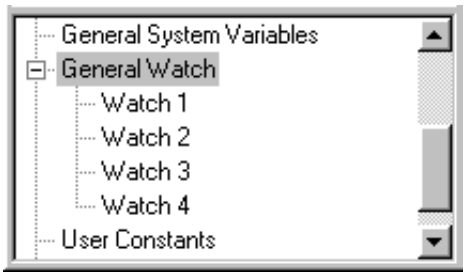


5. Select **OK**. The newly created or edited value appears in the Tag Window.



Using the General Watch Function

You use the General Watch function to group variables, inputs, and outputs that you want to monitor during your testing in a central location. When you open the General Watch function, it appears in the Tag Window. You can have up to four watches available.



To use the General Watch function, select **General Watch** in the Tag Explorer. The title, General Watch, appears at the top of the Tag Window. All variables, inputs and outputs that were previously selected for monitoring by the General Watch function now appear in the Tag Window.

In the following example, you select a variable to be monitored by the General Watch function. The same variable short-cut menu is used for deleting or editing a variable value. Use the same methods for adding any variable to this window for monitoring.

To select a variable for monitoring by the General Watch function:

1. In the Tag Explorer, select any category of variable, input, or output. The values contained in the category you select appear in the Tag Window.
2. In the Tag Window, select a specific variable, input, or output with the right mouse button. The variable short-cut menu appears:

3. Select **Watch.** The selected variable, input or output displays in the General Watch list in the Tag Window, along with general watch items previously selected for monitoring.

User Variables and Constants

The Expression Builder lets you use defined user variables, constants, and local DH-485 variables as elements in expressions. Select User Constants, User Variables, or DH-485 in the Tag Explorer. Then select the desired variable or constant in the Tag Window.

Before a user variable or constant becomes available for use in an expression, you must first define it. If the Tag Window's scrolling list is empty, no variables or constants have been defined.

All user variables and constants are strings of standard ASCII characters that can be entered directly from the keyboard into an expression.

Inputs and Outputs

You can also include defined discrete inputs, discrete input groups, discrete outputs, and discrete output groups as elements in your expression.

In the Tag Explorer, under the desired communications type Flex I/O, (AxisLink I/O, RIO or SLC) select the desired input or output. Then, in the Tag Window, scroll the list until the desired item appears. If the list is empty, no items have been defined for your diagram.

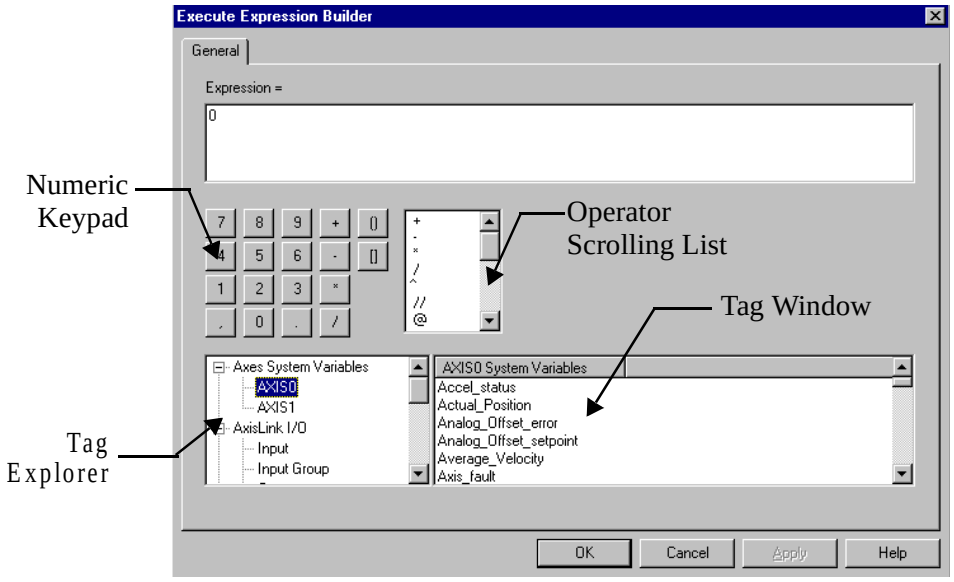
To define a new item:

1. Place the pointing arrow in the Tag Window.
2. Click the right mouse button. A pop-up menu appears.
3. Select **New**. A definition dialog box for the selected item appears.
4. Define the item by completing the dialog box.
5. Select **OK**.

The Expression Builder

The Expression Builder is a specialized calculator for constructing mathematical expressions in GML Commander. You can open the Expression Builder within any GML Commander function block for a field that requires a numeric value or expression.

To access the Expression Builder, place the cursor in the field where you want to create an expression. Click the right mouse button to display the menu. (If the menu doesn't appear, your cursor is not in a field requiring a numeric value.) Select **Build Expression** from the menu to access the Expression Builder dialog box.



Note: The value currently in the field appears in the Expression Builder's Expression = area. The Expression Builder automatically replaces this value with the expression as you create it.

The Expression Builder Dialog Box contains the following features:

- **Numeric Keypad** — To enter numeric values, parentheses, brackets, decimal point, comma, and certain mathematical operators.
- **Operator Scrolling List** — To enter mathematical, logical, and relational expression operators.
- **Tag Explorer —** To select the type of variable, I/O, function or constant and the Tag Window displays.
- **Tag Window** — To select a particular variable, I/O, function or constant.

Calculation Accuracy

In the motion controller, all calculations and intermediate results are kept as double precision floating point values. Calculations and intermediate results can range between 9.46×10^{-308} and 1.79×10^{308}.

The operation of the Expression Builder's various components are explained in detail in the following sections.

The Numeric Keypad

The Expression Builder provides a familiar four-function numeric keypad for entering numbers and the four basic mathematical operations of addition (+), subtraction (-), multiplication (*), and division (/). Select the calculator keys as required to enter numbers and operations into your expression.

Precedence

All expression operators have equal precedence in an expression. This means that expressions are always evaluated from left to right regardless of what operators are used in the expression. This minimizes computation time, when the motion controller evaluates the expression.

For example, the following two expressions do not produce the same result:

The expression 100 * 2 + 10 is evaluated as:

100 * 2 + 10 = 200 + 10 = 210

The expression 2 + 10 * 100 is evaluated as:

2 + 10 * 100 = 12 * 100 = 1,200

( ) *Parentheses*

Using parentheses in an expression changes the default left-to-right precedence of operations. When parentheses are used, the motion controller evaluates an expression from the innermost parentheses outward. Up to 16 levels of parentheses can be used in an expression.

You can insert parentheses into an expression using the **( )** key on the numeric keypad, or by using the keyboard's open and close parenthesis keys. Selecting the numeric keypad's **( )** key inserts both an open and a close parenthesis into the expression and leaves the cursor between them, ready for the parenthetical expression.

Continuing with the previous example, these two expressions do produce the same result—the parentheses force the addition to be performed before the multiplication:

100 * (2 + 10)

(2 + 10) * 100

Both of these expressions are evaluated as:

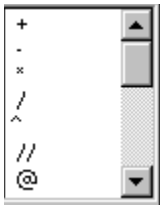
2 + 10 * 100 = 12 * 100 = 1,200

[ ] Brackets

The numeric keypad's bracket key enters an indirect variable reference. See *Indirect Variable in System Functions* in this chapter for more information on indirect variable referencing.

ATTENTION: Do not use the [ ] key in place of the ( ) key to change the precedence of operations in an expression!

Expression Operators



The mathematical and logical operations, which can be performed on the elements (constants, variables, etc.) of an expression, are called expression operators. In general, an expression operator must separate (come between) any two elements in an expression.

Select expression operators from the Operator Scrolling List, shown at left. Scroll the list until the desired symbol appears, then click on it to insert it into the expression at the current cursor location. All expression operators are standard ASCII characters, or combinations of ASCII characters. Therefore, you can also insert them into the expression directly from the keyboard.

There are three kinds of GML Commander expression operators:

- **Mathematical** — Combines the numerical values in an expression.
- **Logical** — Combines the logical (true/false) variables, status conditions, and discrete I/O states to make decisions.
- **Relational** — Lets the numerical and logical comparisons to be made and assigns a value to a variable

Mathematical Operators

Use mathematical operators to combine numerical values in an expression. The available mathematical operators are:

- + (plus sign) — Addition
- - (minus sign) — Subtraction and Unary Minus
- * (asterisk) — Multiplication
- / (slash) — Division
- ^ (carat) —- Exponentiation (to the power of)
- // (double slash) — Integer Division (Quotient)
- @ (at symbol) — Integer Division (Remainder)

Addition +

Use the plus sign to add two elements together. For example, the following expression evaluates to 12.5 if the most recent registration position of Axis 0 is 2.5 position units.

Registration_Position_AXIS0 + 10

Subtraction -

Use the minus sign (or dash) to subtract two elements. For example, the following expression evaluates as 7.5 if the most recent registration position of Axis 0 is 2.5 position units.

Registration_Position_AXIS0 - 10

The minus sign may also be used as a unary minus to negate the value of an element or expression. When used as a unary minus, the minus sign and the element or expression must be enclosed in parentheses. For example, the following expression negates the actual position of Axis 0, resulting in a value of -55.1 if the current actual position of Axis 0 is 55.1 position units.

(-Actual_Position_AXIS0)

Multiplication *

Use the asterisk to multiply two elements together. For example, the following expression evaluates as 3.4 if the current position error of Axis 0 is 6.8 position units.

Position_Error_AXIS0 * 0.5

Division /

Use the slash to divide two elements. For example, the following expression evaluates as 1.66666666666667. Note that the operation performed by the slash is floating point division and the result is accurate to 15 significant digits.

100/60

Exponentiation ^

Use the caret to raise an element to a power. For example, the following expression evaluates as 25. Both the base (5 in the example above) and the exponent (2 in the example above) may be constants, variables, or expressions.

```
5^2
```

// Integer Quotient

The double slash provides the quotient between two elements. It is similar to the single slash floating point divide except that the result is always an integer—all fractional information is truncated. Use Integer Remainder (@) to determine the remainder when two elements are divided (see below). For example, the following expression evaluates as 1.

```
100//60
```

Integer division is useful to calculate the number of times one element can be divided into another. For example, in a feeding application, the number of pieces cut is equal to the quotient of the total material fed and the length of each piece. Thus, the following expression determines the number of pieces cut at any time (assuming that the actual position of Axis 0 was zero when the machine was started).

```
Actual_Position_AXIS0//Cut_Length
```

Specifically, if the current actual position of Axis 0 is 1285.39 position units and the Cut_Length variable is 25, the expression above evaluates to 51 indicating that 51 pieces have been cut.

Integer Remainder @

The @ symbol provides the integer remainder when two elements are divided. It is similar to the double slash integer divide except that it provides only the integer remainder after the division is performed. For example, the following expression evaluates as 40, since 60 goes into 100 once with a remainder of 40.

100@60

The integer remainder is useful to calculate the fractional part when one element is divided into another. For example, in a feeding application, the integer remainder of the total material fed and the length of each piece is the position within the current piece. Thus, the following expression returns the position within the current piece at any time (assuming that the actual position of Axis 0 was zero when the machine was started).

Actual_Position_AXIS0@Cut_Length

Specifically, if the current actual position of Axis 0 is 1285.39 position units and the Cut_Length variable is 25, the expression above evaluates to 10.39 indicating that 10.39 position units of the current piece (piece 52) have been fed.

Logical Operators

Use logical operators to combine logical (true/false) variables to make decisions. Logical operators are divided into bitters operators and Boolean operators. Bitwise operators are used with multiple-bit binary values, such as the value of an input group. Boolean operators are used to combine single-bit logical values and variables.

The logical operators you can use in an expression consist of those shown in the table below.

~ (tilde) — Bitwise NOT (1's Complement)

& (ampersand) — Bitwise AND

| (pipe) — Bitwise OR

! (exclamation point) — Boolean NOT

&& (double ampersands) — Boolean AND

|| (double pipes) — Boolean OR

Bitwise NOT ~

Use the tilde to complement all the bits in a binary value. Technically, this is the 1's complement of the value. It is particularly useful for inverting the value of an input or output group used with negative logic devices'.

ATTENTION: Do not use the Bitwise NOT operator on user or system variables.

For example, the following expression complements the defined bits of the group `Input_Group`.

~Input_Group

Specifically, if `Input_Group` is defined to consist of inputs 2 – 5, which are currently in the following states, the expression above is evaluated as 5 (the complement of $1010_2 = 0101_2 = 5_{10}$):

Input 2 = OFF
Input 3 = ON
Input 4 = OFF
Input 5 = ON

Bitwise AND &

Use the ampersand to logically AND together the corresponding bits in the binary representation of the two elements. It is particularly useful for forcing unwanted bits in an input or output group OFF. This is also known as masking.

ATTENTION: Do not use the Bitwise AND operator on user or system Variables.

For example, if Input_Group is defined to consist of inputs 2 - 5, in the following expression discrete inputs 2 - 5 are logically ANDed with the value 13_{10} (1101_2) when the expression is evaluated.

Input_Group & 13

The result is that discrete input 3 is masked OFF and only inputs 2, 4 and 5 are read. The state of input 3—when ANDed with the 0 at that bit position—always results in a 0 regardless of whether the input is ON or OFF.

Specifically, if inputs 2 - 5 are currently in the following states, the expression above evaluates as 9 (1011_2 & 1101_2 = 1001_2 = 9_{10}).

Input 2 = ON
Input 3 = ON
Input 4 = OFF
Input 5 = ON

Bitwise OR |

Use the pipe (vertical bar) to logically OR together the corresponding bits in the binary representation of the two elements. It is particularly useful for forcing unwanted bits in an input or output group ON.

ATTENTION: Do not use the Bitwise OR operator on user or system variables.

For example, if Input_Group is defined to consist of inputs 2 - 5, in the following expression discrete inputs 2 - 5 are logically ORed with the value 2_{10} (0010_2) when the expression evaluates.

Input_Group | 2

The result is that discrete input 3 is masked ON and only inputs 2, 4 and 5 are read. The state of input 3—when ORed with the 1 at that bit position—always results in a 1 regardless of whether the input is ON or OFF.

Specifically, if inputs 2 - 5 are currently in the following states, the expression above is evaluated as 11 ($1001_2 \mid 0010_2 = 1011_2 = 11_{10}$).

Input 2 = ON
Input 3 = OFF
Input 4 = OFF
Input 5 = ON

Boolean NOT !

Use the exclamation point to complement a single-bit (Boolean) logic element. This is especially useful for forming logical expressions involving status bits, discrete I/O, timers, etc. For example, the following expression has a value of 1(true) if currently there is not a hardware overtravel fault condition on Axis 0 (neither of the overtravel limit switch inputs for Axis 0 is active).

```
!Hardware_overtravel_fault_AXIS0
```

Boolean AND &&

Use the double ampersand to logically AND together two Boolean (single-bit) logic elements. This is especially useful for forming logical expressions involving status bits, discrete I/O, timers, etc. For example, the following expression has a value of 1(true) only when both input 8 and input 9 are ON.

```
Input_8 && Input_9
```

Boolean OR ||

Use the double pipe (vertical bar) to logically OR together two Boolean (single-bit) logic elements. This is especially useful for forming logical expressions involving status bits, discrete I/O, timers, etc. For example, the following expression has a value of 1(true) when there is currently a drive fault condition (drive fault input active) on Axis 0 or Axis 1.

```
Drive_fault_AXIS0 || Drive_fault_AXIS1
```

Relational Operators

Use relational operators to make numerical and logical comparisons, for example, in an On Expression block with If Expression selected. Unlike the other expression operators, that combine elements in an expression, relational operators compare one element to another element or an expression. Therefore, the relational operators are used only if the comparison yields a resultant value of true or false.

The relational operators are shown below:

- = (equal sign) — Equal to
- != (exclamation point with an equal sign) — Not Equal to
- > (right arrow) — Greater Than
- < (left arrow) — Less Than
- >= (right arrow with an equal sign) — Greater Than or Equal to
- <= (left arrow with an equal sign) — Less Than or Equal to

Precedence

All expression operators—including the relational operators—have equal precedence in an expression. This means that expressions are always evaluated from left to right regardless of what operators are used in the expression. This is done to minimize computation time in the motion controller when evaluating the expression.

Because of the left-to-right precedence of operations, when comparing an expression to an element, make sure that the expression is on the left side of the relational operator for proper operation. For example, the following two expressions do not produce the same result.

```
Cut_Length + 10 = Actual_Position_AXIS0
Actual_Position_AXIS0 = Cut_Length + 10
```

The first expression evaluates to 1 (true) if the sum of the cut length variable and 10 equals the actual position of Axis 0. Otherwise it is 0 (false). However, the second expression evaluates to 11, if the actual position of Axis 0 equals the cut length (Actual Position of Axis 0 = Cut Length is true (1); then 1 + 10 = 11) and 10 if not (Actual Position of Axis 0 = Cut Length is false (0); then 0 + 10 = 10).

You can use parentheses to change the default left-to-right precedence of operations, as explained earlier in this section. Continuing the example above, the following two expressions do produce the same result—the parentheses force the addition to be performed before the comparison.

```
Cut_Length + 10 = Actual_Position_AXIS0
Actual_Position_AXIS0 = (Cut_Length + 10)
```

Both expressions evaluate as 1 (true) if the sum of the cut length variable and 10 is equal to the actual position of Axis 0 and 0 (false) otherwise.

Equal to =

Use the equal sign to determine if an element or expression is equal to another element or expression. If the two are equal (have the same value), the comparison is true (1), and if not it is false (0). For example, the following expression has a value of 1 (true) if the current value of the input group Input_Group is 13 and 0 (false) otherwise.

```
Input_Group = 13
```

Not Equal to !

Use an exclamation point in front of an equal sign to determine if an element or expression is not equal to another element or expression. If the two are not equal (do not have the same value), the comparison is true (1). If they are equal, it is false (0). For example, the following expression has a value of 0 (false) if the current value of the input group Input_Group is 13 and 1 (true) otherwise.

```
Input_Group != 13
```

This result is exactly opposite from that obtained when the equal sign is used.

Greater Than >

Use a greater-than sign to determine if one element or expression is greater than another element or expression. If the element or expression on the left side of the greater-than sign has a greater value than the element or expression on the right side, the comparison is true (1). If not, it is false (0). For example, the following expression has a value of 1 (true) if the current position of Axis 0 is greater than the cut length variable, and 0 (false) otherwise.

```
Actual_Position_AXIS0 > Cut_Length
```

Less Than <

Use a less-than sign to determine if one element or expression is less than another element or expression. If the element or expression on the left side of the less-than sign has a lower value than the element or expression on the right side, the comparison is true (1). Otherwise it is false (0). For example, the following expression has a value of 1 (true) if the current position of Axis 0 is less than the cut length variable, and 0 (false) otherwise.

```
Actual_Position_AXIS0 < Cut_Length
```

Greater Than or Equal to >=

Use a greater-than sign followed by an equal sign to determine if one element or expression is greater than or equal to another element or expression. If the element or expression on the left side of the >= is equal to or has a higher value than the element or expression on the right side, the comparison is true (1). Otherwise it is false (0). For example, the following expression has a value of 1 (true) if the current position of Axis 0 is greater than or equal to the cut length variable, and 0 (false) otherwise.

```
Actual_Position_AXIS0 >= Cut_Length
```

Less Than or Equal to <=

Use a less-than sign followed by an equal sign to determine if one element or expression is equal to or less than another element or expression. If the element or expression on the left side of the <= is equal to or has a lower value than the element or expression on the right side, the comparison is true (1). Otherwise it is false (0). For example, the following expression has a value of 1 (true) if the current position of Axis 0 is equal to or less than the cut length variable, and 0 (false) otherwise.

Actual_Position_AXIS0 <= Cut_Length

This result is exactly opposite from that obtained when >= is used.

System Variables

System variables appear in the Expression Builder's Tag Explorer in two groupings:

- **Axis System Variables** — Lets you use motion-related parameter (position, velocity, etc.) and axis status values as elements in expressions.
- **General System Variables** — Lets you use system operation and communications status values as elements in expressions.

Important: Certain system variables depend on the configuration settings made in the Configure Control Options dialog box. These settings determine if they appear in the Tag Window.

To include a system variable in your expression open the Expression Builder and select a system variable type from the Tag Explorer. You can either select Axis System Variables or General System Variables. At the Tag Window, scroll the list and select a variable. The selected variable appears at the cursor position in the Expression = window. Now Complete your expression using other Expression Builder features, and select **OK**.

Because all system variables are strings of standard ASCII characters, you can type them directly into the expression from the keyboard.

The GML Commander system variables are described in the following chapters. For presentation purposes, they are grouped by function into the following groups:

- Motion variables.
- Controller variables.
- Fault variables.
- Status variables.
- Diagnostic variables.
- System functions.
- Mathematical functions.
- User variables.

Motion Variables

Motion variables let you use values, such as axis position and velocity as elements in an expression. All motion variables—except the following—are read-only. They can be used within an expression, but they cannot be assigned a value:

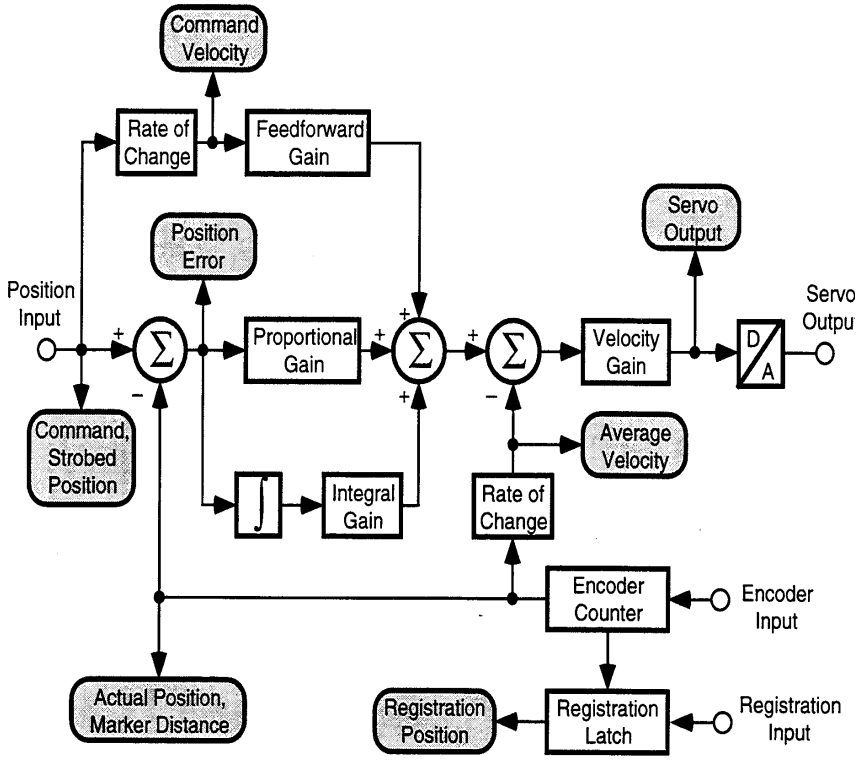
PCAM_good_registration_count
PCAM_bad_registration_count
PCAM_missing_registration_count
PCAM_auto_correction_to_go

The available motion variables are shown in the following table:

Variable	Units	Servo	Master Only	Virtual	Imaginary
Actual_Position	Axis Position Units	✔	✔	✔	
Command_Position	Axis Position Units	✔			✔
Strobed_Position	Axis Position Units	✔	✔	✔	✔
Registration_Position	Axis Position Units	✔	✔	✔	
Soft_Reg_Pos_Axis	Axis Position Units	✔	✔	✔	
Marker_Distance	Axis Position Units	✔	✔	✔	
Distance_To_Go	Axis Position Units	✔			✔
Decel_Distance	Axis Position Units	✔			✔
Encoder_Filter_Lag	Axis Position Units		✔	✔	
Watch_Position	Axis Position Units	✔	✔	✔	✔
Position_Error	Axis Position Units	✔			
Average_Velocity	Axis Pos. Units/Sec.	✔	✔	✔	

Variable	Units	Servo	Master Only	Virtual	Imaginary
Command_Velocity	Axis Pos. Units/Sec.	✔			✔
Servo_Output_Level	Volts	✔			
Servo_Output_Level_mA	mA	✔			
Analog_Offset_setpoint	Volts	✔			
Analog_Offset_error	Volts	✔			
PCAM_auto_correction_to_go	Axis Position Units	✔			✔
PCAM_registration_error	Axis Position Units	✔			✔
PCAM_average_registration_error	Axis Position Units	✔			
PCAM_good_registration_count	Integer	✔			
PCAM_missing_registration_count	Integer	✔			
PCAM_bad_registration_count	Integer	✔			
Axis_iq_reference_1394	%Rated	✔			
Axis_it_limit_1394	%Rated	✔			
Axis_velocity_cmd_1394	RPM	✔			
Axis_torque_cmd_1394	%Rated	✔			

The following figure shows where in the servo loop the motion variables are monitored. In this figure, the gray boxes represent the motion variables and the white boxes represent the servo loop components. Each motion variable is explained following the diagram.

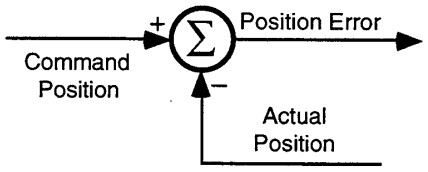


Actual Position

Actual_Position is the current absolute position of a physical or virtual axis (in the position units of that axis) as read from the feedback transducer. In the Tag Explorer, select both Axis System Variables and the desired axis, then select Actual_Position from the Tag Window scrolling list. See the figure above, and Command Position, below, for more information on the relationship among command position, actual position, and position error.

Command Position

Command_Position is the desired or commanded position of a physical axis or the imaginary axis (in the position units of that axis) as generated by any previous motion blocks. In the Tag Explorer, select both Axis System Variables and the desired axis, then select Command_Position from the Tag Window scrolling list.



Actual position is the current position of the axis as measured by the encoder or other feedback device. Position error is the difference between the command and actual positions of a servo axis, and is used to drive the motor to make the actual position equal to the command position.

See your motion controller's *Installation and Setup* manual, for more information about the Nested Digital Servo Loop used in motion controllers.

Command position is useful when performing motion calculations and incremental moves based on the current position of the axis while the axis is moving. Using command position rather than actual position avoids the introduction of cumulative errors due to the position error of the axis at the time the calculation is performed.

Strobed Position

Whenever motion begins for a servo axis (for example, using a Move Axis block), the command position of the axis—at the precise instant the move begins—is stored as the Strobed_Position in the position units of the axis.

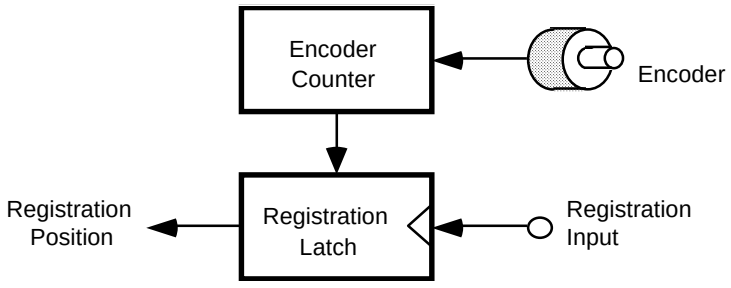
In the Tag Explorer, select both Axis System Variables and the desired axis, then select Strobed_Position from the Tag Window scrolling list. See the figure in Axis System Variables and Command Position earlier in this section for more information on the relationship between command position, actual position, and position error.

Strobed positions are useful to correct for any motion occurring between the detection of an event and the action initiated by the event. For instance, in coil winding applications, strobed position can be used in an expression to compensate for overshooting the end of the bobbin before the gearing direction is reversed. If you know the position of the coil when the gearing direction was supposed to change, and the position at which it actually changed (the strobed position), you can calculate the amount of overshoot, and use it to correct the position of the wire guide relative to the bobbin.

Registration Position

Registration_Position is the absolute position of a physical or virtual axis (in the position units of that axis) at the occurrence of the most recent registration event for that axis. In the Tag Explorer, select both Axis System Variables and the desired axis, then select Registration_Position in the Tag Window.

The figure below shows how the registration position is latched by the registration input when a registration event occurs.



The accuracy of the registration position value, saved as a result of a registration event, is a function of the delay in recognizing the specified transition (1 µs) and the speed of the axis during this time. The uncertainty in the registration position is the distance traveled by the axis during 1 µS (0.000001 second) as shown by the following equation:

$$\text{Uncertainty} = \text{Axis Speed}\left[\frac{\text{Position Units}}{\text{Second}}\right] \times 0.000001 \text{ Seconds}$$

Use the formula above to calculate the maximum registration position error for the expected axis speed. Alternatively, you can calculate the maximum axis speed for a specified registration accuracy by re-arranging this formula as shown in the following equation:

$$\text{Maximum Speed}\left[\frac{\text{Position Units}}{\text{Second}}\right] = \frac{\text{Desired Accuracy } [\text{Position Units}]}{0.000001 \text{ Seconds}}$$

Soft Registration Position

Whenever an axis registration event occurs, the actual position of the axis is latched in hardware and is then available as the Registration_Position variable of the axis. In addition to this hard registration, the actual positions of all physical and virtual axes, as well as the command position of the imaginary axis, are also stored in the appropriate Soft_Reg_Pos*x.x* variables when the registration event occurs. This soft registration stores the positions of all axes as sampled at the beginning of the first servo update following a registration event on any axis.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Soft_Reg_Pos*x.x* in the Tag Window. See *Watch Control*, with Arm Registration selected, in this manual for information on setting up registration events.

There are 42 separate soft registration position variables, one for each of the four physical axes, the two virtual axes, and the imaginary axis. For example, the actual position of Axis 3 is stored in Soft_Reg_Pos_ Axis3_AXIS1, when a registration event occurs on Axis 1. The command position of the imaginary axis is stored in Soft_Reg_Pos_ Imag_AXIS3 when a registration event occurs on Axis 3, etc.

Since the Soft_Reg_Pos*x.x* variables are sampled at the servo update rate rather than directly in hardware, soft registration is not as accurate as hard registration. The position uncertainty of the Soft_Reg_Pos*x.x* variable for an axis is the product of the time between servo updates and the axis speed. Use the formula below to calculate the uncertainty in the software registration position:

$$\text{Soft Registration Position Uncertainty} = \frac{\text{Axis Speed}\left[\frac{\text{Positon Units}}{\text{Second}}\right]}{\text{Servo Update Rate}\,[\text{Hz}]}$$

In the following example, assume a servo update rate of 500 Hz. If Axis 1 is moving at a speed of 2 meters per second and a registration event occurs on Axis 3, the uncertainty of the Soft_Reg_Pos_Axis1_AXIS3 variable is:

$$\text{Soft Registration Position Uncrtainty} = \frac{2\,\frac{\text{Meters}}{\text{Second}}}{500} = 0.004\ \text{m} = 4\ \text{mm}$$

Note that the calculated uncertainty is the maximum error in the direction in which the axis is moving. The error after any individual registration event is somewhere (essentially random) between 0 and the calculated soft registration position uncertainty. In other words,

The soft registration position is:	If:
Greater than the true position of the axis at the occurrence of the registration event	The axis is moving in the positive direction.
Less than the true position	The axis is moving in the negative direction.

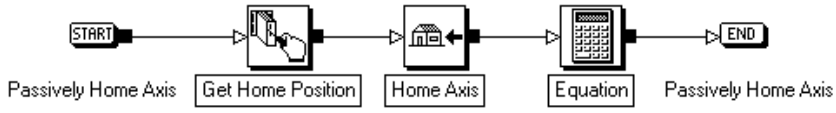
Marker Distance

Marker_Distance is the incremental distance (in axis position units) between the location of the home limit switch, and the encoder marker of a servo, master only or virtual axis as measured during the last (most recent) homing sequence. It is updated whenever the axis is homed to a limit switch and marker using an active homing sequence, or when the axis is passively homed (in this case, the limit switch position is assumed to be 0). Use marker distance when you need to accurately position the encoder marker relative to the home limit switch.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Marker_Distance in the Tag Window. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on axis homing.

You can use the Marker_Distance variable to calculate how much a passive homing operation changes the actual position of the axis. The home limit switch position is assumed to be 0 for passive homing, therefore, Marker_Distance is the position of the axis when the marker occurs—but before it is set to the Home Position. Thus, the change in axis position due to passive homing can be calculated as the Home Position minus the Marker_Distance of the axis. Use a Control Settings block to read the Home Position parameter of the axis.

The Passively Home Axis and Calculate Position Change module included in the GML Commander Module Library under Samples on your GML Commander disk implements this calculation for Axis 0, as shown below.



The Get Home Position block reads the Home Position parameter of Axis 0 and stores it in user variable Home_Position_0. The Home Axis block passively homes the axis and waits for completion.

The Equation block calculates the numerical change in the actual position of the axis caused by the passive homing operation as follows and stores the result in user variable Homing_Change_0.

Marker_Distance_AXIS0 - Home_Position_0

Distance To Go

Distance_To_Go is the remaining incremental distance (in the position units of the axis) left to go in the current move. It is calculated as the difference between the current command position of a servo or the imaginary axis, and the programmed destination for the move. In the Tag Explorer, select both Axis System Variables and the desired axis, then select Distance_To_Go in the Tag Window.

Deceleration Distance

Decel_Distance is the incremental distance (in axis position units) required for a servo or imaginary axis to decelerate to a stop from its current speed using the most recently selected profile. In the Tag Explorer, select both Axis System Variables and the desired axis, then select Decel_Distance in the Tag Window.

Important: The Deceleration distance axis system variable does not apply to jogs.

Encoder Filter Lag

Encoder_Filter_Lag is used to observe the effect of the Encoder Input Filter. It reports the existing filter-induced lag between the filtered output and axis Actual Position. A positive (+) value indicates leading, a negative (-) value indicates lagging.

Note: This variable appears in the Tag Window only if Master Only Axis Type is selected in the General Page of the Configure Axis Use dialog box, and Encoder Filter is selected in the Feedback Page of that dialog box.

Watch Position

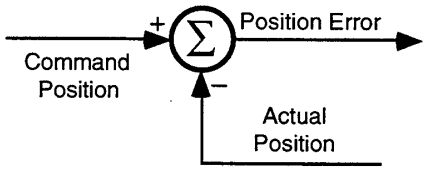
Watch_Position is the current setpoint position of a physical axis or the imaginary axis (in axis position units), as set up in the last (most recent) Watch Control block (with Arm Watch Position selected) for that axis.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Watch_Position in the Tag Window.

Position Error

Position_Error is the difference (in axis position units) between the command and actual positions of a servo axis. In the Tag Explorer, select both Axis System Variables and the desired axis, then select Position_Error in the Tag Window.

Position error is used to drive the motor to make the actual position equal to the command position. The figure below shows the relationship of these three positions.



Average Velocity

Average_Velocity is the current speed of a servo, master only or virtual axis. It is measured in axis position units per second, and calculated by averaging the actual velocity of the axis over the averaged velocity timebase for that axis (entered in the motion controller's application setup menu). Average velocity is always a positive value regardless of which direction the axis is moving.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Average_Velocity in the Tag Window.

The resolution of the Average_Velocity variable is determined by the current value of the Averaged Velocity Timebase parameter, and the conversion constant K (feedback counts per position unit) for the axis. The greater the averaged velocity timebase value, the better the speed resolution, but the slower the response to changes in speed.

The average velocity resolution in position units per second may be calculated using the equation below.

$$\frac{1}{\text{Averaged Velocity Timebase [Seconds]} \times K \left[\frac{\text{Feedback Counts}}{\text{Position Unit}}\right]}$$

The minimum averaged velocity timebase value is one millisecond. Values less than this default to 1 ms. See the *Setup* section of your motion controller's *Installation and Setup* manual for information on setting the averaged velocity timebase and the conversion constant parameters.

For example, on an axis with position units of inches and a conversion constant (K) of 20000, an averaged velocity timebase of 0.25 seconds results in an average velocity resolution of

$$\frac{1}{0.25 \times 20000} = 0.0002 \frac{\text{Inches}}{\text{Second}} = 0.012 \frac{\text{Inches}}{\text{Minute}}$$

The average velocity of the imaginary axis is always zero, because the imaginary axis has no actual position and thus no actual velocity to average. Use command velocity for the imaginary axis.

Command Velocity

Command_Velocity is the desired or commanded speed of a servo axis or the imaginary axis in axis position units per second, as generated by any previous motion blocks. It is calculated as the rate of command position change with time. Unlike average velocity, however, command velocity is a signed value—the sign (+ or -) depends on which direction the axis is moving.

In the Tag Explorer select both Axis System Variables and the desired axis, then select Command_Velocity in the Tag Window. See the figure in this chapter for more information on command velocity.

Command_Velocity is a signed floating-point value with up to 15 significant digits. Its resolution does not depend on the averaged velocity timebase, or the conversion constant of the axis.

Servo Output Level

Servo_Output_Level is the current voltage level of the servo output of a servo axis. For the Compact, Servo_Output_Level_mA is the current output level of the servo output of a servo axis configured for current output. Use Servo_Output_Level if the servo output for the axis is configured for voltage (±10V), and Servo_Output_Level_mA if it is configured for current (±150 mA).

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Servo_Output_Level in the Tag Window.

Analog Offset Setpoint

Analog_Offset_setpoint is the analog value, set in an Analog Offset function block's *Setpoint* field, which the motion controller sets and holds.

In the Tag Explorer select both Axis System Variables and the desired axis, then select Analog_Offset_setpoint in the Tag Window.

Analog Offset Error

Analog_Offset_error is the position error between the selected axis' actual analog position, and the setpoint that was input in an Analog Offset function block's *Setpoint* field.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Analog_Offset_error in the Tag Window.

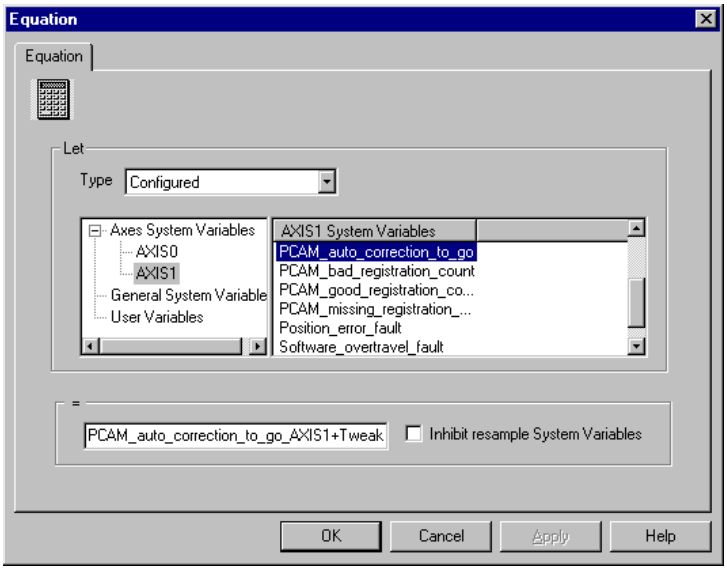
PCAM Auto Correction Distance To Go

PCAM_auto_correction_to_go is the auto-correction indexer's remaining correction distance to go, for a position-lock cam slave axis with auto-correction enabled.

In the Tag Explorer, select both Axis System Variables and the desired servo axis, then select PCAM_auto_correction_to_go in the Tag Window.

Although you must select a slave axis in the Tag Explorer, the PCAM_auto_correction_to_go value is returned as a signed (±) value in position units of the corresponding position-lock cam master axis. The sign of the returned value indicates the direction in which the slave axis is currently correcting.

Unlike other motion variables, you can change PCAM_auto_correction_to_go using an equation block. This is useful to add a small manual correction to an auto-correcting position-lock cam while it is running. The following Equation block adds the value of the user variable Tweak to the position-lock cam auto-correction distance to go for Axis 1.



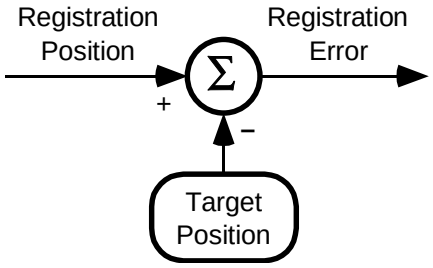
PCAM Registration Error

PCAM_registration_error is used by the auto-correction feature of position-lock cams to automatically correct for error between the measured registration position, and the desired or expected registration position.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select PCAM_registration_error in the Tag Window. Although you must select a slave axis in the Tag Explorer, the PCAM_registration_error value is returned in position units of the corresponding:

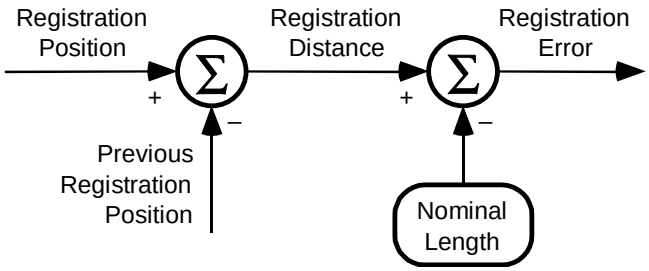
The PCAM_registration_error value is returned in position units of the corresponding:	**For:**
Master axis of the position-lock cam	Absolute or relative auto-correction.
Registering axis	Absolute ratioed or relative ratioed auto-correction.

The calculation of the PCAM_registration_error value differs depending on the type of auto-correction used. For absolute and absolute ratioed auto-correction, the registration error is the difference between the most recent hard or soft registration position of the axis, and the target position as shown below.



The registration error is returned in the position units of the corresponding master axis of the position-lock cam for absolute auto-correction, and in the position units of the corresponding registering axis for absolute ratioed auto-correction.

For relative and relative ratioed auto-correction, the registration error is the difference between measured registration distance of the axis and the nominal length, as shown below.



The registration distance is the difference between the most recent hard or soft registration position of the axis, and the previous registration position. The registration error is returned in the position units of the corresponding master axis of the position-lock cam for relative auto-correction, and in the position units of the corresponding registering axis for relative ratioed auto-correction.

PCAM Average Registration Error

PCAM_average_registration_error is the arithmetic average of the last ten registration event's PCAM_registration_error values, for a position-lock cam slave axis with auto-correction enabled.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select PCAM_average_registration_error in the Tag Window.

Although you must select a slave axis in the Tag Explorer:

The PCAM_average_registration_error value is returned in position units of the corresponding:	**For:**
Master axis of the position-lock cam	Absolute or relative auto-correction.
Registering axis	Absolute ratioed or relative ratioed auto-correction.

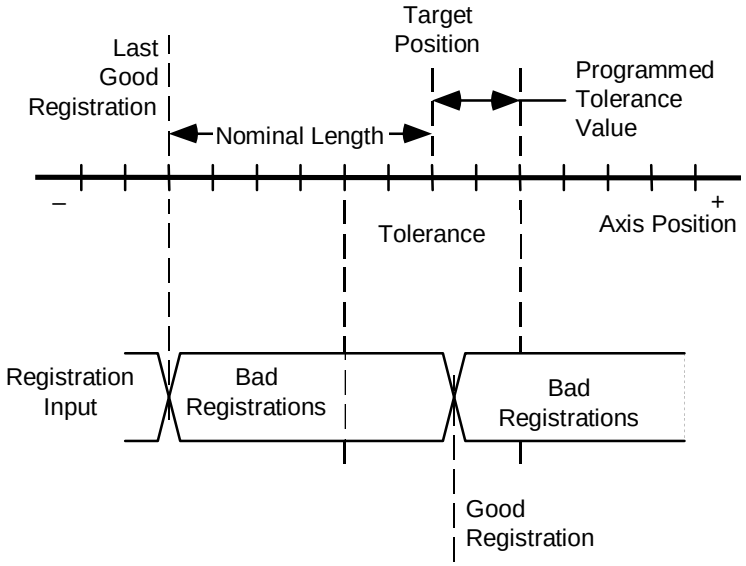
PCAM Good Registration Count

PCAM_good_registration_count is the current number of consecutive in-tolerance registration events, which have occurred on the registering axis for a position-lock cam with auto-correction enabled.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select PCAM_good_registration_count in the Tag Window.

PCAM_good_registration_count is set to zero when power is applied to the motion controller, or when the position-lock cam for the specified slave axis is disabled. You can reset this variable to zero (or any other value) by assigning it a value using an Equation block.

An in-tolerance registration event is one that occurs inside the target position or nominal length tolerance entered in the Configure Cam block, as shown below.



PCAM_good_registration_count is incremented when an in-tolerance registration event occurs. As shown above, the first registration event that occurs within the specified tolerance is considered the good registration. All registration events that occur after a good registration event, and before the beginning of the next tolerance area, are considered bad—even if they occur within the current tolerance. If you did not select Tolerance in the Configure CAM block's Auto Correction page, all registration events on the specified axis are considered good.

PCAM Missing Registration Count

PCAM_missing_registration_count is the current number of consecutively missed registration events that have occurred on the registering axis for a position-lock cam with Auto-Correction enabled.

In the Tag Explorer select both Axis System Variables and the desired axis, then select PCAM_missing_registration_count in the Tag Window.

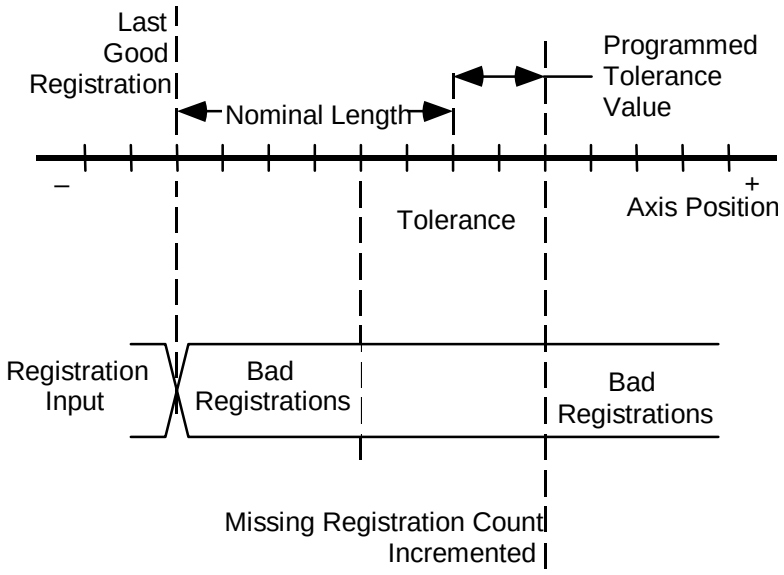
A registration event is deemed to be missing if either of the following is true:

- the position-lock cam master axis (relative and absolute)
- the registering axis (relative ratioed and absolute ratioed)

moves one of the following:

- a distance equal to the specified nominal length plus the specified tolerance past the last in-tolerance registration event (relative and relative ratioed)
- past the position specified by the absolute target position plus the specified tolerance (absolute and absolute ratioed) without the occurrence of another registration event

The following illustration demonstrates missing registration for relative auto-correction.



PCAM_missing_registration_count is incremented when a registration event is missed, and is reset to zero when the next good (in-tolerance) registration event occurs.

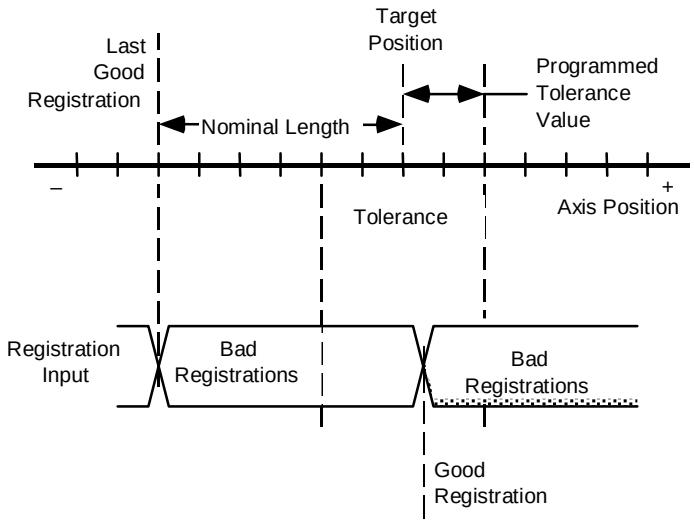
PCAM_missing_registration_count is set to zero when power is applied to the motion controller. It can also be reset to zero (or any other value) using an equation block.

Bad Registration Count

PCAM_bad_registration_count is the current number of consecutive out-of-tolerance registration events, which have occurred on the registering axis for a position-lock cam with auto-correction enabled.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select PCAM_bad_registration_count in the Tag Window.

An out-of-tolerance registration event is one that occurs outside the target position or nominal length tolerance entered in the Configure Cam block, as shown below.



PCAM_bad_registration_count is incremented when an out-of-tolerance registration event occurs. It is reset to zero when an in-tolerance registration event occurs. As shown above, all registration events, which occur after a good registration event and before the beginning of the next tolerance area, are considered bad—even if they occur within the current tolerance.

PCAM_bad_registration_count is set to zero when power is applied to the motion controller. It can be reset to zero (or any other value) using the equation block.

Axis Iq Reference 1394

The Axis_iq_reference_1394 variable reports the instantaneous torque/current that the controller commands to the designated axis module, as a percent of rated current.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Axis_iq_reference_1394 in the Tag Window.

Axis It Limit 1394

The Axis_it_limit_1394 variable reports the current value of the computed It Limit for the designated axis, as a percent of rated current. This value changes dynamically as the commanded current to the axis is integrated over time. In situations demanding high duty cycle acceleration and deceleration motion profiles, the Drive It Limit can actually fall below the Axis I limit setting in order to protect the drive from an It thermal fault condition.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Axis_it_limit_1394 in the Tag Window.

Axis Velocity Command 1394

The Axis_velocity_cmd_1394 variable reports the current velocity value being commanded by the 1394 controller for the designated axis, with the servo loop type set for velocity mode.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Axis_velocity_cmd_1394 in the Tag Window.

Axis Torque Command 1394

The Axis_torque_cmd_1394 variable reports the current torque value being commanded by the 1394 controller for the designated axis, when the servo loop type is set for torque mode.

In the Tag Explorer, select both Axis System Variables and the desired axis, then select Axis_torque_cmd_1394 in the Tag Window.

Controller Variables

Controller variables let you use timer values and states, analog input values, the last character received from the operator interface, and the current task number as elements in an expression. These controller variables are all general system variables:

Variable	Units
Analog_Input_0	Volts
Analog_Input_1	Volts
Analog_Input_2	Volts
Analog_Input_3	Volts
Free_Running_Clock	Seconds
Timer1	Seconds
Timer2	Seconds
Timer3	Seconds
Timer4	Seconds
Last_Keypress	Decimal ASCII Value
Current_Task	Integer (0 to 9)
CPU_utilitzation	Normalized Percent
CPU_utilization_peak	Normalized Percent
Controller_Address	Integer (0 to 7)
AxisLink_configuration_nodes	Integer (0 to 255)

Each of these variables is explained below.

Analog Inputs

Four analog inputs are available as an option on IMC-S/20x-A and IMC-21x-A model motion controllers for reading the values from analog transducers, etc. Analog input values range between ±10 volts, depending on the input level.

In the Tag Explorer, select General System Variables, then select Analog_Input_0-3 in the Tag Window.

If you selected iCODE version 2.4, or earlier, in the Configure Control Options dialog box, the motion controller can read voltage on each analog input at any time by using the appropriate analog input variable. IMC S Class Compact (IMC-S/23x) motion controllers use Flex I/O to provide analog inputs.

If you selected iCODE version 3.0, or later, in the Configure Control Options dialog box, you must select Flex I/O in the Tag Explorer to select defined analog inputs in the Tag Window. See *Inputs and Outputs* for more information.

See the *Online Help* for more information on configuring and defining Flex I/O analog inputs.

Free Running Clock

The free running clock is a count-up timer that runs continuously whenever the motion controller is powered. Its values are displayed as seconds. You can use the Set Timer block to set the free running clock to a value, and you can read the free running clock's value using the Free_Running_Clock variable.

In the Tag Explorer, select General System Variables, then select Free_Running_Clock in the Tag Window.

Use the free running clock for timing the length of events. Se

Free_Running_Clock is a double precision floating point value with a practical maximum limit of 15 significant digits.

Timers

Four count-down timers are provided for programming dwells and other time-dependent functions. You can set each timer to a value using the Set Timer block, and read its value (in seconds) at any time using the appropriate timer variable.

In the Tag Explorer, select General System Variables, then select Timer1-4 in the Tag Window.

You can use timer variables as mathematical variables or as logic variables in an expression. In a mathematical expression, timer variables read the time remaining on the timer. In a logical expression, the timer value is 1 (true) after the timer has been set and while it is counting down. When the timer times out (reaches zero), both the mathematical value and the logical value of the timer variable is 0 (false).

Last Keypress (Character)

Last_Keypress is the decimal ASCII value of the last (most recent) character (key) received by the operator interface serial port. Ordinarily, you will use the On Key Press function block's If Key and Wait for Key selections to check the last keypress variable, instead of using the Last_Keypress variable directly in an expression.

In the Tag Explorer, select General System Variables, then select Last_Keypress in the Tag Window.

Current Task

Current_Task is the task number of the currently executing task when multitasking is enabled. It is always an integer value, from 0 to 9 inclusive. If you are not multitasking, Current_Task is 0.

In the Tag Explorer, select General System Variables, then select Current_Task in the Tag Window.

Use multitasking blocks to enable and disable multitasking and to start and stop individual tasks.

Use the Current_Task variable when several tasks use identical (duplicate) modules in a diagram. Within a duplicate module, you can use the Current_Task variable to make decisions based on which task is currently executing the module.

CPU Utilization

CPU_utilization is a floating-point value, between 0 and 1, which represents the relative amount of the available CPU capacity currently being used for motion-related calculations.

In the Tag Explorer, select General System Variables, then select CPU_utilitzation in the Tag Window.

You can use the CPU_utilization variable during application development to check whether program execution and RIO Adapter block transfer speed are being degraded due to the application's motion requirements. Subtracting the CPU_utilization from 1 yields the relative amount of CPU capacity available for processing the application program (generated from the GML Commander diagram and downloaded to the motion controller). See the *CPU Utilization* chapter for more information.

CPU Utilization Peak

The CPU_utilization_peak variable (enabled only in iCODE versions 3.5 and higher) displays the historic high value for the CPU_utilization variable. This value is reset at power up, and when you press the motion controller's Reset button.

In the Tag Explorer, select General System Variables, then select CPU_utilitzation_peak in the Tag Window.

Controller Address

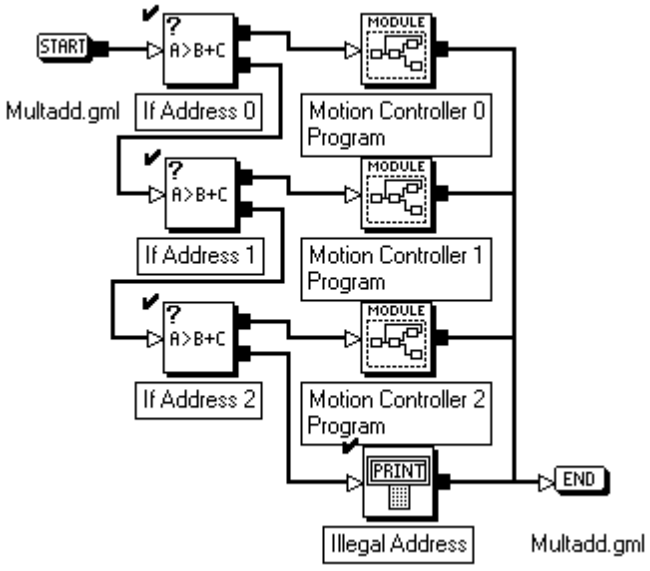
If Extended AxisLink is not selected in the General page of the Configure Control Options dialog box, Controller_address is an integer value from 0 to 7, and equals the current setting of the motion controller's front panel ADDRESS selector switch.

If Extended AxisLink is selected in the General page of the Configure Control Options dialog box, Controller_address is an integer value from 0 to 15 and is set by the software using in the AxisLink page of the Configure Control Options dialog box.

In the Tag Explorer, select General System Variables, then select Controller_address in the Tag Window.

Use the Controller_address variable to let the application program read the selected address of the motion controller, in which the application program is running, and configure itself accordingly.

For example, the GML Commander diagram shown below checks the motion controller address and then executes the module containing the appropriate program for this motion controller. If the address is not legal, a message is printed and the program ends.



This GML Commander diagram is included on the GML Commander distribution disk in the Samples directory as MULTADD.GML.

AxisLink Configuration Nodes

AxisLink_configuration_nodes is an eight-bit binary coded value that indicates which external controllers are connected via AxisLink.

In the Tag Explorer, select General System Variables, then select AxisLink_configuration_nodes in the Tag Window.

You can use this parameter to check which nodes are on the link. When used in conjunction with the default data parameter AxisLink_nodes (D63), it can check which, if any, controllers have dropped off the link. The AxisLink_nodes parameter contains a bit mapping of the controllers that are supposed to be linked, whereas the AxisLink_configuration_nodes parameter contains a bit mapping of the controllers that are in fact currently linked.

For example, if Controller 0 is currently linked to Controllers 1, 3, 5, and 7:

- bits 1, 3, 5, and 7 will be set in AxisLink_configuration_nodes
- the value 170 (AA hex) will be displayed when AxisLink_configuration_nodes is examined.

Important:

ATTENTION: If you selected Extended AxisLink in the Configure Control Options dialog box, you can select from up to 16 controllers to link together. However, only eight controllers can be linked at any one time

Fault Variables

Fault variables let you use fault conditions as elements in an expression. Fault variables include numerical representations of global faults, axis faults, and fault codes, as well as logical variables for individual faults.

The following table describes upper level—or general—fault variables. Later sections describe both the fault code values and the lower tier fault variables associated with the following general fault variables.

Variable	Units
Global_fault	Integer from 0 to 16
Runtime_fault	Integer from 0 to 35
Axis_fault	Integer from 0 to 7
Axis_bus_loss_fault_1394	0 (False) or 1 (True)
Axis_drive_over_temp_fault_1394	0 (False) or 1 (True)
Axis_it_fault_1394	0 (False) or 1 (True)
Axis_module_hard_fault_1394	0 (False) or 1 (True)
Axis_Motor_over_temp_fault_1394	0 (False) or 1 (True)
Axis_power_fault_1394	0 (False) or 1 (True)
AxisLink_failed	0 (False) or 1 (True)
AxisLink_fault_code	Integer from 0 to 128
AxisLink_general_fault	0 (False) or 1 (True)
AxisLink_timeout	0 (False) or 1 (True)
CNET_fault	0 (False) or 1 (True)
CNET_fault_code	Integer from 0 to 3
Drive_fault	0 (False) or 1 (True)

Variable	Units
Drive_hard_fault_1394	0 (False) or 1 (True)
DH485_fault_code	Integer from 0 to 4
DH485_general_fault	0 (False) or 1 (True)
DSP_feedback_fault_1394	0 (False) or 1 (True)
Encoder_noise_fault	0 (False) or 1 (True)
Encoder_loss_fault	0 (False) or 1 (True)
FLEX_fault	0 (False) or 1 (True)
FLEX_fault_code	Integer from 0 to 7
Hardware_overtravel_fault	0 (False) or 1 (True)
Position_error_fault	0 (False) or 1 (True)
Resolver_loss_fault_1394	0 (False) or 1 (True)
RIO_fault_code	Integer from 0 to 9
SLC_fault	0 (False) or 1 (True)
SLC_fault_code	Integer from 0 to 3
Software_overtravel_fault	0 (False) or 1 (True)
System_bus_over_voltage_flt_1394	0 (False) or 1 (True)
System_bus_undr_voltage_flt_1394	0 (False) or 1 (True)
System_control_power_fault_1394	0 (False) or 1 (True)
System_ground_fault_1394	0 (False) or 1 (True)
System_module_hard_fault_1394	0 (False) or 1 (True)
System_over_temp_fault_1394	0 (False) or 1 (True)
System_phase_loss_fault_1394	0 (False) or 1 (True)
System_serial_ring_fault_1394	0 (False) or 1 (True)
System_smrt_pwr_i_limit_fault_1394	0 (False) or 1 (True)
System_smrt_pwr_pre_charge_fault_1394	0 (False) or 1 (True)

Variable	Units
System_smrt_pwr_shunt_timeout_fault_1394	0 (False) or 1 (True)

Each of the upper-tier, general fault variables along with all related lower tier variables and fault codes are explained in the following pages.

Global Fault

Global_fault is an integer value representing the highest priority fault currently active in the motion controller. Global faults include controller memory faults as well as any faults on any axis. The table below displays the global fault value alongside the corresponding message that appears in a global fault field in the runtime display (if enabled).

Fault Value	Description	Runtime Display
16	CNET_fault	CNET FLT
15	CPU_Utilization_Overrun_Fault	CPU FLT
14	Feedback_Device_Fault	FDB FLT
13	Flex_Fault	FLX FLT
12	SLC_Fault	CPU FLT
11	DH-485_General_Fault	DH FLT
10	RIO_Fault	RIO FLT
9	AxisLink _Timeout_Fault or AxisLink_Failed on any virtual axis	AXL FLT
8	AxisLink_General_Fault	AXL FLT
7	Setup_Data_Memory_Fault	MEM FLT
6	Application_Program_Memory_Fault	PRG FLT
5	Drive_Fault on any axis	DRV FLT
4	Position_Error_Fault on any axis	ERR FLT
3	Hardware_Overtravel_Fault on any axis	HRD LIM

Fault Value	Description	Runtime Display
2	Software_Overtravel_Fault on any axis	SFT LIM
1	Encoder_Loss_Fault or Encoder_Noise_Fault on any axis	ENC FLT
0	No Faults	AXES OK

Global faults are prioritized from highest to lowest, in the order shown in the table above. When multiple faults are active, the Global_fault value represents the highest priority fault. For instance, if a hardware overtravel fault is active for an axis, and a software travel limits exceeded condition is also active on another axis, Global_fault has a value of 3. When the highest priority fault is cleared, Global_fault has the value of the next highest priority fault.

You use an End Program block, with both When End or Fault and Go to When End if Global Fault Occurs selected, to intercept global faults that occur when the application program is running. The When End or Fault sequence should evaluate the Global_fault variable, correct the problem, reset the specific fault variable to zero to clear the fault, and restart the program or the tasks as required.

CNET Fault

CNET_fault is a logical (Boolean) variable, which has a value of:

- 1 (true) if a CNET error has occurred, and
- 0 (false) if not.

When CNET_fault = 1, Global_fault = 16.

Refer to the CNET_fault_code variable later in this chapter for a description of the particular fault that occurred.

Refer to online help for the particular CNET_fault_code value for help in diagnosing and clearing the fault.

This variable applies only to 1394 Turbo controllers.

CPU Utilization Overrun Fault

The motion controllers continuously monitor average CPU utilization for performance and safety purposes. A CPU utilization overrun (Global_fault = 15) indicates that the average CPU utilization has exceeded 90%. When a CPU utilization overrun fault occurs, the controller:

- stops all motion on all axes
- disables feedback on all axes
- sets all servo outputs to zero
- deactivates all drive enables
- disables CPU Watchdog output

The occurrence of this type of fault indicates that the servo update rate is set to a value that is too high to support the complexity of the application running in the controller.

The motion controller displays the CPU Utilization Overrun fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

Feedback Device Fault

The motion controllers periodically perform integrity checking of the position feedback interface chips installed in the system. If any of these chips fail one of the checks, a feedback device fault (Global_fault = 14) indicates the device failure. If such a condition occurs, Encoder_Loss_Fault (Resolver_Loss_Fault on the 1394 GMC/1394 GMC Turbo) is also indicated on the axes that are associated with the failed device. A feedback device fault usually requires a service replacement.

The motion controller displays the Feedback Device Fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

Flex Fault

A Flex I/O Device fault applies to Compact or 1394 GMC/1394 GMC Turbo controllers configured with Flex I/O modules. A Flex I/O device fault (Global_fault = 13) means a fault has occurred when the controller tried to communicate with one or more of the Flex I/O blocks. When a flex I/O device fault occurs, the FLEX_fault_code system variable is set to the first flex I/O module detected with a communication problem. This fault usually indicates a hardware (e.g., cabling, power, or device failure) problem.

The motion controller displays the Flex I/O Device Fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

SLC Fault

On motion controllers with the SLC interface option, an SLC processor fault (Global_fault = 12) indicates that a power failure or other error has occurred within the SLC interface. Refer to SLC Fault Code for more information.

The motion controller displays the SLC Processor Fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

When global_fault = 12, an SLC fault has occurred. SLC_fault is a logical (Boolean) variable which has values of:

- 1 (true) if a power failure has occurred within the SLC interface
- 0 (false) if not

In the Tag Explorer select General System Variables, then select SLC_fault in the Tag Window.

Refer to the SLC_fault_code variable for information regarding the fault. Once you have corrected the problem, clear it by assigning a value of 0 to the SLC_fault variable using an Equation block. When the SLC_fault variable is set to 0, the variable SLC_fault_code is automatically reset to 0. For more information, refer to the *1394 User Manual* (publication 1394-5.0).

DH-485 General Fault

On Compact or 1394 GMC/1394 GMC Turbo controllers with DH-485 enabled, a DH-485 fault (Global_fault = 11) indicates that a communication or data transfer error has occurred on the DH-485 network interface. Refer to DH-485 Fault Code for more information.

The motion controller displays the DH-485 Fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

DH485_general_fault is a logical (Boolean) variable. It applies to IMC S Class Compact or 1394 GMC/1394 GMC Turbo controllers with DH-485 enabled, and has a value of:

- 1 (true) a communication or data transfer error has occurred on the DH-485 network interface, and
- 0 (false) if not

When DH-485_general_fault = 1, Global_fault 11. The motion controller also displays the fault in the Terminal Window or the runtime display (if enabled).

You can directly clear a DH-485 general fault using a Reset DH-485 Fault block. See *DH-485 Fault Code* for more information on the conditions that cause this fault and how to clear them.

RIO Fault

On motion controllers with the RIO option (IMC-S/2xx-R models or 1394-SJTxx-C-RL), an RIO fault (Global_fault = 10) indicates a fault in the RIO option. See RIO Fault Code for more information on the specific conditions that cause this fault.

The motion controller displays the RIO Fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

AxisLink Timeout Fault

On motion controllers with the AxisLink option (IMC-S/2xx-L models or 1394-SJTxx-C-RL), an AxisLink virtual axis fault (Global_fault = 9) indicates that the current AxisLink virtual axis connection has failed or could not be established. See AxisLink Timeout and AxisLink Failed for more information on the specific conditions that cause this fault.

The motion controller displays the AxisLink Virtual Axis Fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

AxisLink General Fault

For motion controllers with the AxisLink option (1394-SJTxx-C-RL or IMC-S/2xx-L models), an AxisLink general fault (Global_fault = 8) indicates an AxisLink timeout occurred, when the controller attempted to access another motion controller's AxisLink outputs or data. See AxisLink General Fault and AxisLink Fault Code for more information on the specific conditions that cause this fault.

In the Tag Explorer, select General System Variables, then select AxisLink_general_fault in the Tag Window.

Setup Data Memory Fault

The motion controllers use a checksum to verify the integrity of the power-up setup parameter values in memory. Whenever the setup values are changed via the motion controller's setup menus, a new checksum is calculated and stored with the data. A setup data memory fault (Global_fault = 7) indicates that the stored checksum is not the same as the current checksum and that memory has been corrupted.

If a setup data memory fault occurs—indicating corruption of the setup parameters—verify all setup parameter values by going through the setup menus in the motion controller. See *Setup* in the *Installation and Setup* manual for your motion controller for information on using the setup menus.

Important: To avoid future data corruption, be sure to lock the memory using the front panel keyswitch or the memory unlock jumper, after re-entering the setup values.

The motion controller displays the setup data memory fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

Application Program Memory Fault

The motion controllers use a checksum to verify the integrity of the application program (created from a GML Commander diagram) stored in memory. Whenever you download a new program to the motion controller, a new checksum is calculated and stored with the program. An application program memory fault (Global_fault = 6) indicates that the stored checksum is not the same as the current checksum and that memory has been corrupted.

If an application program memory fault occurs—indicating corruption of the application program—determine the source of this discrepancy and correct it, then re-download the application program. After re-downloading, be sure to lock the memory using the front panel keyswitch or the memory unlock jumper to avoid future data corruption.

The motion controller displays the application program memory fault in the Terminal Window or the runtime display (if enabled). However, this fault variable is not available in the Expression Builder.

Axis Global Faults

The remaining global fault values indicate a fault on one of the motion controller axes:

- Drive Fault (Global_fault = 5)
- Position Error Tolerance Exceeded (Global_fault = 4)
- Hardware Overtravel Fault (Global_fault = 3)
- Software Travel Limits Exceeded (Global_fault = 2)
- Encoder Noise or Loss Fault (Global_fault = 1)

See *Axis Fault* for descriptions of the individual axis faults.

Axis Fault

Axis_fault is an integer value representing the highest priority fault currently active on a specific axis. The table below shows the axis fault values and the corresponding message that appears in a global fault or axis status field in the terminal window or the runtime display (if enabled).

Fault Value	Description	Runtime Display	Servo	Master Only	Virtual	Imaginary
7	AxisLink Timeout	AXL FLT			✔	
6	AxisLink Failed (axis not found)	AXL FLT			✔	
5	Drive Fault / Motor Thermal 1394	DRV FLT	✔			
4	Position Error Tolerance Fault	ERR FLT	✔			
3	Hardware Overtravel Fault	HRD LIM	✔			
2	Software Overtravel Fault	SFT LIM	✔			
1	Encoder Noise or Loss Fault	ENC FLT	✔	✔		
0	No Faults	AXIS OK	✔	✔	✔	✔

The axis faults are prioritized from highest to lowest in the order shown in the table above. This means that when a given fault is active, another fault of lower priority may also be active. For instance, if a hardware overtravel fault is active for an axis, a software travel limits exceeded condition may also be active on the axis. When the highest priority fault is cleared, Axis_fault has the value of the next highest priority fault.

See the descriptions of the individual axis faults below for an explanation of these faults.

AxisLink Timeout

AxisLink_timeout is a logical (Boolean) variable that has a value of:

- 1 (true) if an AxisLink virtual axis connection which had been operating properly has timed out, or

- 0 (false) if not

An AxisLink timeout is defined to have occurred when the linked axis has failed to provide any new information for at least 5 servo update intervals. The AxisLink LED on the front panel flashes red when an AxisLink timeout occurs.

When AxisLink_timeout = 1, Axis_fault = 7, Axis_status = 14, and Global_fault = 9.

You can directly clear an AxisLink timeout on a virtual axis by using a Reset Fault block or by assigning a value of 0 to the appropriate AxisLink_timeout variable using an Equation block. However, this does not automatically re-enable the virtual axis.

AxisLink Failed

AxisLink_failed is a logical (Boolean) variable which has values of

- 1 (true) if an AxisLink virtual axis connection failed to be established (the axis could not be found)
- 0 (false) if not

The AxisLink LED on the front panel flashes green when an AxisLink failure occurs.

In the Tag Explorer, select both Axis System Variables and the desired virtual axis (an AxisLink failed can only occur on AxisLink virtual axes), then select AxisLink_failed in the Tag Window.

When evaluated in an expression, the value of the AxisLink_failed variable indicates whether or not an attempted AxisLink connection failed.

When AxisLink_failed = 1, Axis_fault = 6, Axis_status = 13, and Global_fault = 9.

You can directly clear an AxisLink failure on a virtual axis using a Reset Fault block or by assigning a value of 0 to the appropriate AxisLink_failed variable using an Equation block. However, this does not automatically re-enable the virtual axis.

Drive Fault / Motor Thermal 1394

Drive_Fault applies to Compact, Integrated, and Basic controllers.

Motor_Thermal_1394 applies to 1394 controllers. Although the Terminal Window or other display will continue to display Drive_Fault, even for 1394 controllers, the fault is actually a Motor_Thermal_1394 fault. The information about Drive_Fault also applies to the Motor_Thermal_1394 fault.

Drive_fault is a logical (Boolean) variable which has values of:

- 1 (true) if the drive fault input for the specified axis was activated
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Drive_fault in the Tag Window.

When evaluated in an expression, the value of the Drive_fault variable indicates whether a drive fault has occurred.

When Drive_fault = 1, Axis_fault = 5, Axis_status = 11, and Global_fault = 5 if no other faults of higher priority are active.

A Drive_fault can only occur if the Compact, Integrated, or Basic motion controller is configured to use the drive fault input.

A Motor_Thermal_1394 fault can only occur if the 1394 controller is configured to use the motor thermal input.

See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the drive fault and motor thermal inputs.

When configured to use the drive fault input, each physical axis of the motion controller can be further configured to respond to a drive fault in different ways. If drive fault action is set to STOP MOTION, then when the drive fault input is activated, the axis immediately decelerates to a stop without disabling feedback or the drive enable output. If DISABLE DRIVE is selected, when the drive fault input is activated, axis feedback is immediately disabled, the servo amplifier output is zeroed, and the appropriate drive enable output is deactivated.

If drive fault action is set to STATUS ONLY, drive faults must be handled within the GML Commander diagram. In general, this setting should only be used in applications where neither the standard STOP MOTION nor KILL DRIVE actions are appropriate. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the drive fault action.

You can directly clear a drive fault on an axis by assigning a value of 0 to the appropriate Drive_fault variable using an Equation block. However, this does not automatically re-enable feedback or the drive, and if the drive fault input is still activated after executing the Equation block, the fault occurs again immediately.

A drive fault is also cleared when a Feedback On or Home Axis block is executed, or when the setup menus in the motion controller are run. However, this does not automatically re-enable feedback or the drive. See *Setup* in the *Installation and Setup* manual for your motion controller for more information on the setup menus.

Position Error Fault

Position_error_fault is a logical (Boolean) variable which has values of:

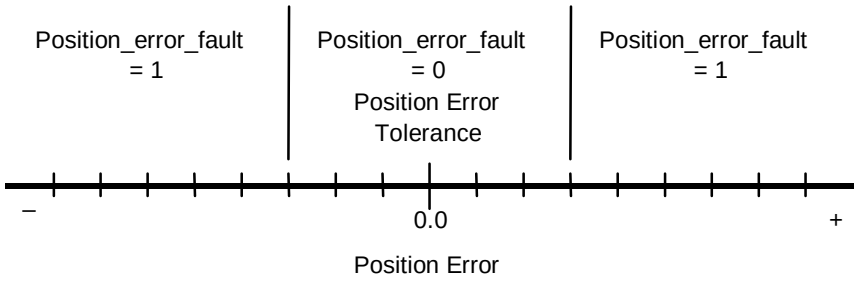
- 1 (true) if the position error tolerance for the specified axis has been exceeded
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Position_error_fault in the Tag Window.

When evaluated in an expression, the value of the Position_error_fault variable indicates whether a position error fault has occurred.

When Position_error_fault = 1, Axis_fault = 4, Axis_status = 10 if no other faults of higher priority are active on the axis, and Global_fault = 4 if no other faults of higher priority are active on any axis. A position error fault is usually caused by commanding a speed or acceleration that exceeds the capability of the drive system, a jammed or stuck axis, or a faulty feedback device.

Position error tolerance is the amount of position error (±) the motion controller tolerates without causing a position error fault.



See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on position error tolerance.

Each physical axis of the motion controller can be configured to respond to a position error fault in different ways. If position error fault action is set to STOP MOTION, then, when the position error tolerance is exceeded, the axis immediately decelerates to a stop without disabling feedback or the drive enable output. If DISABLE DRIVE is selected, when the position error tolerance is exceeded, axis feedback is immediately disabled, the servo amplifier output is zeroed, and the appropriate drive enable output is deactivated.

If position error fault action is set to STATUS ONLY, position error faults must be handled within the GML Commander diagram. In general, this setting should only be used in applications where neither the standard STOP MOTION nor DISABLE DRIVE actions are appropriate. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the position error fault action.

You can directly clear a position error fault on an axis by assigning a value of 0 to the appropriate Position_error_fault variable using an Equation block. However, this does not automatically re-enable feedback or the drive. If the position error of the axis is still greater than the position error tolerance, after executing the Equation block, the fault occurs again immediately. A position error fault is also cleared when a Feedback On or Home Axis block is executed, or when the setup menus in the motion controller are run. See *Setup* in the *Installation and Setup* manual for your motion controller for more information on the setup menus.

Hardware Overtravel Fault

Hardware_overtravel_fault (sometimes called Hardware Travel Limits Fault) is a logical (Boolean) variable which has values of:

- 1 (true) if either the positive or negative overtravel input for the specified axis was activated
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Hardware_overtravel_fault in the Tag Window.

When evaluated in an expression, the value of the Hardware_overtravel_fault variable indicates whether or not a hardware overtravel fault has occurred on a specific axis.

When Hardware_overtravel_fault = 1, Axis_fault = 3, Axis_status = 9 if no other faults of higher priority are active on the axis, and Global_fault = 3 if no other faults of higher priority are active on any axis.

A hardware overtravel fault can only occur if the motion controller is configured to use overtravel limit switches. See in the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the overtravel inputs.

When configured to use overtravel limit switches, each physical axis of the motion controller can be further configured to respond to a hardware overtravel fault in different ways. If hard overtravel fault action is set to STOP MOTION, then when either of the overtravel inputs is activated, the axis immediately decelerates at maximum deceleration to a stop without disabling feedback or the drive enable output. If DISABLE DRIVE is selected, when either of the overtravel inputs is activated, axis feedback is immediately disabled, the servo amplifier output is zeroed, and the appropriate drive enable output is deactivated.

If hard overtravel fault action is set to STATUS ONLY, hardware overtravel faults must be handled within the GML Commander diagram. This setting should only be used in applications where neither the standard STOP MOTION nor DISABLE DRIVE actions are appropriate. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the hard overtravel fault action.

To clear a hardware overtravel fault:

1. Use the Control Settings block to adjust Hardware Overtravel Checking (data bit #4) by disabling it.
2. Use the Reset Fault block, set to **Axis Fault**, to clear the Hardware Overtravel fault.
3. Use the Feedback block to enable feedback for the axis.
4. Use the Move Axis block to move the axis to a point that does not trigger either a hardware or a software overtravel fault.

ATTENTION: Be careful to move the axis in the direction opposite the previous move (i.e. away from the overtravel condition). Movement in the wrong direction may cause physical harm to materials, machinery or to the operator.

5. Use the Control Settings block to re-enable the Hardware Overtravel Checking data bit.

Software Overtravel Fault

Software_overtravel_fault (sometimes called Software Travel Limits Fault) is a logical (Boolean) variable which has values of:

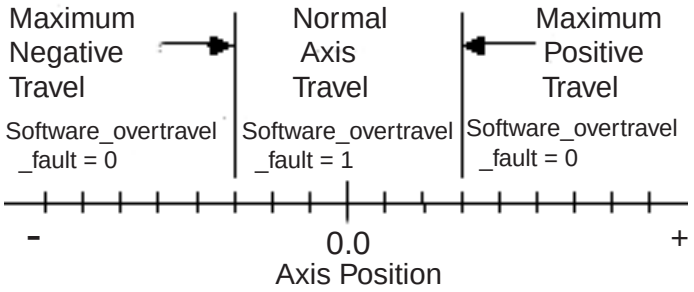
- 1 (true) if either the maximum positive or maximum negative travel for the specified axis is exceeded
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Software_overtravel_fault in the Tag Window.

When evaluated in an expression, the value of the Software_overtravel_fault variable indicates whether or not the position of a specific axis has exceeded either the maximum positive or maximum negative travel parameter values.

When Software_overtravel_fault = 1, Axis_fault = 2, Axis_status = 8 if no other faults of higher priority are active on the axis, and Global_fault = 2 if no other faults of higher priority are active on any axis.

A software overtravel fault can only occur if the motion controller is configured to use software travel limits. See in the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring software travel limits.



When configured to use soft travel limits, each physical axis of the motion controller can be further configured to respond to a software travel fault in different ways. If soft overtravel fault action is set to STOP MOTION, then when either of the travel limits is exceeded, the axis immediately decelerates at maximum deceleration to a stop without disabling feedback or the drive enable output. If DISABLE DRIVE is selected, when either of the travel limits is exceeded, axis feedback is immediately disabled, the servo amplifier output is zeroed, and the appropriate drive enable output is deactivated.

If soft overtravel fault action is set to STATUS ONLY, software overtravel faults must be handled within the GML Commander diagram. This setting should only be used in applications where neither the standard STOP MOTION nor DISABLE DRIVE actions are appropriate. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the soft overtravel fault action.

To clear a software overtravel fault:

1. Use the Control Settings block to adjust and disable Software Overtravel Checking (data bit #6).
2. Use the Reset Fault block, set to **Axis Fault**, to clear the Software Overtravel fault.
3. Use the Feedback block to enable feedback for the axis.
4. Use the Move Axis block to move the axis to a point that does not trigger either a hardware or a software overtravel.

ATTENTION: Be careful to move the axis in the direction opposite the previous move (i.e. away from the overtravel condition). Movement in the wrong direction may cause physical harm to materials, machinery or to the operator.

5. Use the Control Settings block to re-enable the Software Overtravel Checking data bit.

Encoder Noise Fault

Encoder_noise_fault is a logical (Boolean) variable which has values of:

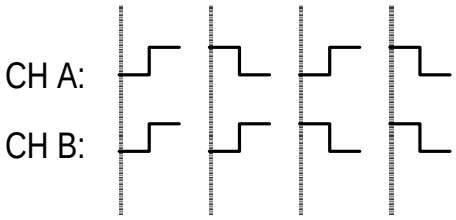
- 1 (true) if encoder noise has been detected on the specified axis
- 0 (false) if not

Note: An Encoder is sometimes alternately referred to as a Transducer. Thus, Encoder Noise Fault is sometimes referred to as Transducer Noise Fault.

In the Tag Explorer, select both Axis System Variables and the desired physical axis then select Encoder_noise_fault in the Tag Window.

When evaluated in an expression, the value of the Encoder_noise_fault variable indicates whether encoder noise has been detected on the encoder input for the axis. When Encoder_noise_fault = 1, Axis_fault = 1, Axis_status = 7 if no other faults of higher priority are active on the axis, and Global_fault = 1 if no other faults of higher priority are active on any axis.

Encoder noise is a simultaneous transition on both encoder channels.



Each physical axis of the motion controller can be configured to respond to encoder noise in different ways. If the encoder noise fault action is set to STOP MOTION, then when encoder noise is detected, the axis immediately decelerates to a stop at maximum deceleration without disabling feedback or the drive enable output. If DISABLE DRIVE is selected, when encoder noise is detected, axis feedback is immediately disabled, the servo amplifier output is zeroed, and the appropriate drive enable output is deactivated.

If the transducer noise fault action is set to STATUS ONLY, encoder noise faults must be handled within the GML Commander diagram. In general, this is the recommended setting since encoder noise faults should be cured at machine commissioning. See in the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the encoder noise fault action.

An encoder noise fault on an axis can be directly cleared by assigning a value of 0 to the appropriate Encoder_noise_fault variable using an Equation block. However, this does not automatically re-enable feedback or the drive. An encoder noise fault is also cleared when a Feedback On or Home Axis block is executed or when the Setup Menus in the motion controller are run. See *Setup* in the *Installation and Setup* manual for your motion controller for more information on the Setup Menus.

Encoder Loss Fault

Encoder_loss_fault is a logical (Boolean) variable which has values of:

- 1 (true) if the encoder signal is lost on the specified axis

- 0 (false) if not

Note: An Encoder is sometimes alternately referred to as a Transducer. Thus, Encoder Loss Fault is sometimes referred to as Transducer Loss Fault

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Encoder_loss_fault in the Tag Window.

When evaluated in an expression, the value of the Encoder_loss_fault variable indicates whether encoder signal loss has been detected on the encoder input for the axis.

When Encoder_loss_fault = 1, Axis_fault = 1, Axis_status = 7 if no other faults of higher priority are active on the axis, and Global_fault = 14 if no other faults of higher priority are active on any axis.

Encoder loss fault is indicated if either of the following occur:

- Both of the differential signals for any encoder channel (A+ and A-, B+ and B-, or Z+ and Z-) are at the same level (both low). Under normal operation, the differential signals are always at opposite levels. The most common cause of this situation is a broken wire between the encoder and the motion controller.
- Loss of encoder power or encoder common to the motion controller or the encoder.

Each physical axis of the motion controller can be configured to respond to encoder loss in different ways. If the transducer loss fault action is set to DISABLE DRIVE, when transducer loss is detected, axis feedback is immediately disabled, the servo amplifier output is zeroed, and the appropriate drive enable output is deactivated. If the transducer loss fault action is set to STATUS ONLY, encoder loss faults must be handled within the GML Commander diagram. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the encoder loss fault action.

An encoder loss fault on an axis can be directly cleared by assigning a value of 0 to the appropriate Encoder_loss_fault variable using an Equation block. However, this does not automatically re-enable feedback or the drive, and if encoder signals are still lost after executing the Equation block, the fault occurs again immediately. An encoder loss fault is also cleared when a Feedback On or Home Axis block is executed or when the setup menus in the motion controller are run. See *Setup* in the *Installation and Setup* manual for your motion controller for more information on the setup menus.

Flex Fault

Flex_fault applies to IMC S Class Compact or 1394 GMC/1394 GMC Turbo controllers configured with Flex I/O modules. It is a logical (Boolean) variable which has values of:

- 1 (true) if there has been a communications failure with a Flex I/O module
- 0 (false) if not

If Flex_fault = 1, Global_fault = 13. When this fault occurs, the Flex_fault_code system variable displays the address of the first flex I/O module detected with a communication problem. The motion controller displays the Flex I/O Device Fault in the Terminal Window or the runtime display (if enabled).

Flex Fault Code

Flex_fault_code is an integer value, from 0 to 7, representing the first detected failed Flex I/O module.

For example, if Flex_fault_code = 3 and Flex_fault = 1, the 4^{th} Configured Flex Module should be checked for defects.

In the Tag Explorer select General System Variables, then select Flex_fault_code in the Tag Window.

AxisLink General Fault

AxisLink_general_fault is a logical (Boolean) variable which has values of:

- 1 (true) if there has been an AxisLink timeout attempting to access another motion controller's AxisLink outputs or data
- 0 (false) if not

In the Tag Explorer select General System Variables, then select AxisLink_general_fault in the Tag Window.

When evaluated in an expression, the value of the AxisLink_general_fault variable indicates whether a fault has occurred attempting to access another motion controller's AxisLink outputs or data.

When AxisLink_general_fault = 1, Global_fault = 8 if no other faults of higher priority are active on any axis.

You can directly clear an AxisLink general fault by using a Clear AxisLink Fault block, or assigning a value of 0 to the AxisLink_general_fault variable using an Equation block. However, this does not automatically re-enable the AxisLink connection to the other motion controller. See *AxisLink Fault Code* for more information on the conditions that cause this fault and how to clear them.

AxisLink Fault Code

AxisLink_fault_code is an integer value representing the highest priority condition causing an AxisLink general fault, as shown in the table below.

In the Tag Explorer select General System Variables, then select AxisLink_fault_code in the Tag Window.

Fault Code	Description	LED Color
128	Offline (≥ 256 Link Errors)	Red
96	Failing (≥ 96 Link Errors)	Flashing R/G
79	Timeout Accessing Controller 15 Data	Flashing Red
78	Timeout Accessing Controller 14 Data	Flashing Red
77	Timeout Accessing Controller 13 Data	Flashing Red

Fault Code	Description	LED Color
76	Timeout Accessing Controller 12 Data	Flashing Red
75	Timeout Accessing Controller 11 Data	Flashing Red
74	Timeout Accessing Controller 10 Data	Flashing Red
73	Timeout Accessing Controller 9 Data	Flashing Red
72	Timeout Accessing Controller 8 Data	Flashing Red
71	Timeout Accessing Controller 7 Data	Flashing Red
70	Timeout Accessing Controller 6 Data	Flashing Red
69	Timeout Accessing Controller 5 Data	Flashing Red
68	Timeout Accessing Controller 4 Data	Flashing Red
67	Timeout Accessing Controller 3 Data	Flashing Red
66	Timeout Accessing Controller 2 Data	Flashing Red
65	Timeout Accessing Controller 1 Data	Flashing Red
64	Timeout Accessing Controller 0 Data	Flashing Red
47	Timeout Accessing Controller 15 Outputs	Flashing Red
46	Timeout Accessing Controller 14 Outputs	Flashing Red
45	Timeout Accessing Controller 13 Outputs	Flashing Red
44	Timeout Accessing Controller 12 Outputs	Flashing Red
43	Timeout Accessing Controller 11 Outputs	Flashing Red
42	Timeout Accessing Controller 10 Outputs	Flashing Red
41	Timeout Accessing Controller 9 Outputs	Flashing Red
40	Timeout Accessing Controller 8 Outputs	Flashing Red
39	Timeout Accessing Controller 7 Outputs	Flashing Red
38	Timeout Accessing Controller 6 Outputs	Flashing Red
37	Timeout Accessing Controller 5 Outputs	Flashing Red
36	Timeout Accessing Controller 4 Outputs	Flashing Red

Fault Code	Description	LED Color
35	Timeout Accessing Controller 3 Outputs	Flashing Red
34	Timeout Accessing Controller 2 Outputs	Flashing Red
33	Timeout Accessing Controller 1 Outputs	Flashing Red
32	Timeout Accessing Controller 0 Outputs	Flashing Red
31	Link Error Accessing Controller 15	Flashing Green
30	Link Error Accessing Controller 14	Flashing Green
29	Link Error Accessing Controller 13	Flashing Green
28	Link Error Accessing Controller 12	Flashing Green
27	Link Error Accessing Controller 11	Flashing Green
26	Link Error Accessing Controller 10	Flashing Green
25	Link Error Accessing Controller 9	Flashing Green
24	Link Error Accessing Controller 8	Flashing Green
23	Link Error Accessing Controller 7	Flashing Green
22	Link Error Accessing Controller 6	Flashing Green
21	Link Error Accessing Controller 5	Flashing Green
20	Link Error Accessing Controller 4	Flashing Green
19	Link Error Accessing Controller 3	Flashing Green
18	Link Error Accessing Controller 2	Flashing Green
17	Link Error Accessing Controller 1	Flashing Green
16	Link Error Accessing Controller 0	Flashing Green
0	No AxisLink General Fault	Green

AxisLink general faults are prioritized from highest to lowest in the order shown in the table above. A timeout accessing another controller's data has higher priority than a timeout accessing another controller's AxisLink outputs. This means that when multiple faults are active, the AxisLink_fault_code value represents the highest priority fault on any

controller. When the highest priority fault is cleared, AxisLink_fault_code has the value of the next highest priority fault.

Offline

The offline condition (AxisLink_fault_code = 128) indicates that there have been at least 256 link errors and that AxisLink has shut down. This is usually caused by a cabling or power-up sequence problem. If this condition occurs, the AxisLink LED is red and the motion controller must be reset or AxisLink disabled and then re-enabled using the AxisLink data bit. See Control Settings in the Function Blocks chapter for more information on setting and resetting data bits.

Failing

The failing condition (AxisLink_fault_code = 96) indicates that there have been at least 96 link errors and that AxisLink is in danger of going offline. This is usually caused by an intermittent cabling problem, excessive noise, improper cable termination, or a failing AxisLink module. In this condition, the AxisLink LED flashes green and red and the link continues to operate, but it could fail completely (offline) at any time.

If the application program is not running, the motion controller attempts to re-establish the failing links. Thus, it is possible that the failing condition will correct itself (No AxisLink General Fault) if the source of the problem is corrected in time. If the application program is running, the controller does not attempt to re-establish the failing links—this must be done by the fault handling routine in the program.

Timeout Accessing Data

The timeout accessing controller data conditions (AxisLink_ fault_code = 64 - 79) indicate that a Read Remote Value block in the GML Commander diagram was not able to read the specified value from the specified motion controller (more than 20 milliseconds have elapsed with no response from the specified motion controller). The AxisLink LED flashes red when this condition occurs. Check that there is a motion controller connected to AxisLink with the address specified in the Read Remote Value block, and that it is working properly with AxisLink enabled.

Timeout Accessing Output

The timeout accessing controller AxisLink outputs conditions (AxisLink_fault_code = 32 - 47) indicate that the specified motion controller has stopped updating its AxisLink outputs (more than 1 second has elapsed with no response from the specified motion controller). The AxisLink LED flashes red when this condition occurs. Check that there is a motion controller connected to AxisLink with the address specified in the AxisLink Inputs or AxisLink Input Groups definition, and that it is working properly with AxisLink enabled.

Error Accessing Controller

The link error accessing controller conditions (AxisLink_fault_ code = 16 - 31) indicate that a link error occurred while attempting to establish a link with the specified motion controller to read its AxisLink outputs. This is usually caused by an intermittent cabling problem, excessive noise, improper cable termination, or a failed AxisLink module. The AxisLink LED flashes green when this condition occurs. Check that there is a motion controller connected to AxisLink with the specified address, and that it is working properly with AxisLink enabled.

Runtime Fault

Runtime_fault is an integer value representing the last (most recent) illegal condition detected in the motion controller while the application program was running, as shown in the following table.

In the Tag Explorer select General System Variables, then select Runtime_fault_code in the Tag Window.

Fault Value	Description
35	Stack Fault
34	Fault Ring Error 1394
33	Commission Error 1394
32	DSP Error 1394
31	Illegal Merge Attempt (Interpolation)

Fault Value	Description
30	Flex I/O Missing or Failed
29	Illegal Path Reference
28	Insufficient Time (Linear Interpolation)
27	Bad Arc (Circular Interpolation)
26	Unknown RIO Device (Scanner)
25	Attempt to Access Unknown AxisLink Device
24	Optional Hardware Missing
23	No Tasks Running
22	Attempt to Access Locked Memory
21	Illegal Direct Command
20	Program Checksum Error
19	Illegal Command While Program Running
18	Illegal Command While in Overtravel
17	Illegal Command with Feedback OFF
16	Illegal Command with Feedback ON
15	Value Out of Range
14	Illegal Data Reference
13	Illegal Axis Configuration
12	Illegal Expression
11	Illegal Action
10	Illegal Numeric Format
9	Illegal Axis Reference
8	Extra Characters in Command
7	Missing { or } in Command
6	Missing (or) in Command

Fault Value	Description
5	Missing [or] in Command
4	Missing Label for Goto
3	Missing = in Expression
2	Missing Comma (,) in Command
1	Illegal Command
0	No Fault Detected

You use an End Program block, configured for When End of Fault with Go to When End if Runtime Fault Occurs selected, to intercept runtime faults, which occur while the application program is running. The When End or Fault sequence should evaluate the Runtime_fault variable, correct the problem, set Runtime_fault to zero to clear the fault, and restart the program or the tasks as required.

If the diagram contains an End Program block without Go to When End if Runtime Fault selected, or if an End Program block is not included in the diagram, the program aborts when a runtime fault occurs. You then can use the Show Program Status block from the Online Manager to check the Runtime_fault variable and determine which fault occurred.

Stack Fault

A Stack fault (Runtime_fault = 35) occurs when the “program stack” (an assigned portion of working memory, or DRAM) no longer accurately directs the flow of program execution.

Note: Stack Fault applies to iCODE versions 3.0 and higher.

The program stack contains references (or pointers) to locations in the application program iCODE script. Each time program execution “jumps” to another, non-sequential part of the iCODE script a pointer to the location in the iCODE - from which program execution “jumped” - is stored in the program stack. This pointer is used to return program execution to the original point of departure, when execution of the non-sequential code is complete.

The number of pointers a program stack can contain ("program stack depth") varies by controller, as follows:

Controller	Call Depth
IMC S Class 1394 Turbo	40
IMC S Class 1394	25
IMC S Class Compact	25

The Stack Fault occurs when:

- the number of program calls to the stack exceeds the program stack depth, or
- the number of returns exceeds the number of calls from the iCODE script to the program stack.

The effect of this fault on program execution is the same as issuing a Stop command. When this fault occurs:

- program execution halts,
- motion on all axes stops at 100% of Maximum Deceleration, and
- Feedback remains enabled.

GML Commander displays a message identifying the name of the function block and the module that triggered the fault.

To correct the problem:

1. Check the block and module referenced in the error message.
2. If the referenced block precedes a Native Code block, the problem may be iCODE corruption. If so, correct any obvious iCODE errors in the Native Code block.

3. Download a “clean” version of the application program, and re-run the program. If the fault does not recur, the problem probably was a stack underflow, and should be solved.

4. If the block referenced in the error message includes an ordinary call to another part of the program, the problem is probably a stack overflow. In this case, simplify your program by eliminating stack calls, so the program stack depth of your controller is not exceeded.

5. Consider running your program in either Trace or Step mode, or after setting breakpoints, to let you follow execution of your program. This may reveal more calls - especially repetitive calls - than you might have anticipated.

6. After you have removed enough calls, re-download the application program, reset the controller and re-execute.

7. If the problem recurs, return to step 1, above.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Fault Ring Error 1394

A Fault Ring Error (Runtime_fault = 34) is detected if:

- All System, Axis and DIM modules are not properly plugged in,
- The connecting pins on any module are bent,
- The Terminator is not properly installed, or
- A hardware problem exists.

To correct the problem:

Verify that each element of the 1394 fault ring (the system module, the axis modules, any DIM modules and the terminator) is completely attached. To do so, press together both ends of the fault ring, thereby fully engaging the connecting pins.

If the problem persists, its cause lies within the hardware. Isolate the malfunctioning module, as follows:

1. Remove all axis and DIM modules.
2. Plug the terminator into the system module. If the system module is OK (system LED flashing red/green), go to step 3. If not (system LED solid red) go to step 4.
3. Plug in each axis and DIM module one at a time, re-attaching the terminator after each module is added. See which module causes the system to fail (system LED solid red).

Note: If no module fails, combine the axis modules in different order, until one fails.

4. Return the malfunctioning module to the manufacturer for repair.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Commission Error 1394

Some known causes for a Commission Error 1394 (Runtime_fault = 33), and suggested responses, are listed below.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Cause	Comment/Response
Initialization of the drive(s) and motor(s) failed.	Verify that the Motor ID setting in the Motor/Drive page of the Configure Axis Use dialog box identifies the motor now being used.
The Reset Fault block - set to Reset 1394 - was issued as a direct command while the program is running.	The Reset Fault block - set to Reset 1394 ñ can be included in a diagram, and executed as a direct command when program execution is stopped, but cannot be executed as a direct command during program execution.

Cause	Comment/Response
A system or axis module hard fault occurred.	To check for a hard fault: 1. Select General System Variables in the Tag Explorer, and scroll to Drive_Hard_Fault_1394 in the Tag Window. 2. Check for the presence of an Axis_module_hard_fault_ 1394 or a System_module_hard_fault_ 1394. 3. If either fault exists, find the underlying axis or system hard fault. 4. Locate and remedy the cause of the fault. 5. Clear the underlying fault with a Reset Fault block set to Reset 1394. 6. Press the Reset button on the controller.
A hardware problem exists with a system, axis, or DIM module.	Isolate the malfunctioning module, as follows: 1. all axis and DIM modules. 2. Plug the terminator into the system module. If the system module is OK (system LED flashing red/green), go to step 3. If not (system LED solid red) go to step 4. 3. Plug in each axis and DIM module one at a time, re-attaching the terminator after each module is added. See which module causes the system to fail (system LED solid red). **Note:** If no module fails, combine the axis modules in different order until one fails. 4. Return the malfunctioning module to the manufacturer for repair.

DSP Error 1394

A DSP Error 1394 (Runtime_fault = 32) indicates that an internal hardware error has occurred within the DSP in the system module. Consequently, handshaking between the i960 and the DSP stops during firmware code execution.

To correct the problem:

Press the Reset button on the system module. If the problem persists, return the controller to the manufacturer for repair.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Illegal Merge Attempt (Interpolation)

An attempted merge of two Interpolate Axes block moves has failed (Runtime_fault = 31), because the selected interpolator:

- is accelerating,
- is decelerating, or
- has another merge pending.

To correct the problem:

Edit the application program to eliminate any merge of interpolated moves occurring under these circumstances.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Flex I/O Missing or Failed

This fault (Runtime_fault = 30) indicates that a timeout occurred while waiting for the serial bus to respond. This condition can be caused by:

- a failure to detect the Flex I/O, or
- a hardware problem within the controller.

To correct the problem:

1. Verify that all modules are receiving 24 volts of direct current as required.

2. Verify that all Flex I/O modules are installed on their rack in the same order that they are:
   - enabled in the Flex I/O page of the Configure Control Options dialog box, and
   - configured in the Configure Flex I/O Module menu.
3. Restart the controller.

If the problem persists, it is a hardware problem. Return the controller to the manufacturer for repair.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Insufficient Time (Linear Interpolation)

his fault (Runtime_fault = 28) indicates that a timed linear interpolated move (set in an Interpolate Axes block with Trapezoidal Profile selected) failed because:

- the calculated Accel/Decel exceeds the input Accel/Decel, or
- the calculated velocity exceeds the input speed.

Consequently, there is not enough time to move the commanded distance at the input Speed and Accel/Decel.

To correct the problem:

Increase one or more or the following timed linear interpolated move settings:

- Time
- Speed
- Accel/Decel.

See the online help topic Timed Linear Interpolation for more information.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Bad Arc (Circular Interpolation)

This fault (Runtime_fault = 27) occurs when an Interpolate Axes block executes in either of two instances.

First, with Intermediate Arc selected, either:

- the current position, the end point and the intermediate point are all on the same line, or
- two of these points are identical.

Second, with Radius Arc selected, the arc endpoint does not lie on the circle (i.e. within 0.5%) described by the starting point and center point.

To correct the problem:

Check your calculations to insure that no two points are the same and no two points are on the same line (for Intermediate Arcs), and that the endpoint lies on the described circle (for Radius Arcs).

See the online help topic Radius Arc – Bad Arcs for more information.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Attempt to Access Unknown AxisLink Device

This fault (Runtime_fault = 25) indicates that communications between two AxisLinked controllers has failed, because the local controller is either:

- not receiving data from the intended remote controller, or
- receiving data from an unintended remote controller.

This occurs when the local controller requests flag information from another AxisLink node, which information has not been updated and is no longer valid.

To correct the problem:

- Check the diagram for complete configuration of all AxisLink-related settings (e.g. AxisLink I/O Input and Input Group dialog boxes, Read Remote Value block, etc.)
- Check the cable and terminators connecting each node (controller) on the AxisLink.
- If all else fails, there may be a hardware problem with the AxisLink card. If so, return the controller to the manufacturer for repair.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

No Tasks Running

This fault (Runtime_fault = 23) indicates that the application program is running, but no task is executing. Its cause can be:

- a Task Control block, with Stop Current Task or Stop Other Task selected, has stopped all tasks; or
- a hardware problem internal to the controller.

To correct the problem:

Check the use of Task Control blocks in the application program. Be sure that execution of all tasks is not stopped simultaneously. See the Task Control block Stop Current Task and Stop Other Task selections.

If the problem persists, it is a hardware problem. Return the controller to the manufacturer for repair.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Attempt to Access Locked Memory

This fault (Runtime_fault = 22) indicates that, because the Memory switch (on the front of the controller) is locked, any one of the following Terminal Window data-transfer iCODE commands cannot be executed:

- start new program (!N)
- set a power-up data bit (BS)
- set a power-up data parameter (DS)
- transfer a diagram checksum (X)
- initialize setups (.I), or
- upload working values (.U).

To correct the problem:

1. Set the Memory switch to unlocked.
2. Repeat the data transfer iCODE command.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Illegal Direct Command

This fault (Runtime_fault = 21) indicates one of the following conditions:

- Because AxisLink is not enabled, the AxisLink Debug Terminal Window iCODE direct command (B) cannot be executed; or,
- Because no program is currently running, the following Terminal Window iCODE direct commands cannot be executed:
  - call a sub-routine (#)
  - return from a subroutine (;)
  - repeat loop (<)
  - go to (>)
  - end program (E).

To correct the problem:

Either:

- Enable AxisLink in the General and AxisLink pages of the Configure Control Options dialog box; or
- Start program execution using the Go command in either the Online Toolbar or the Diagram menu.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Illegal Command While Program Running

This fault (Runtime_fault = 19) indicates that, because a program is currently running, the following Terminal Window direct iCODE commands cannot be executed:

- list program (!L)
- new program (!N)
- run program (!R)
- access servo setup menu (.G) ñ (where this is not part of the program)
- access hookup diagnostics (.H)
- access runset menu (.R)
- access setup menu (.S) ñ (where this is not part of the program)
- encoder filter debug test (BE)

To correct the problem:

1. Stop execution of the program.
2. Re-execute the Terminal Window iCODE command.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Illegal Command While In Overtravel

This fault (Runtime_fault = 18) indicates that no command can be executed because:

- a jog or a move (in either the positive or the negative direction) has caused a Hardware_overtravel_fault or a Software_overtravel_fault, and
- the system response (set in the Fault Action page of the Configure Axis Use dialog box) is to Disable Drive or Stop Motion.

To correct the problem:

1. Clear the overtravel condition. This can include some or all of the following steps:
   - Reset the controller and turn feedback on, using a Feedback block (for Disable Control fault action).
   - Turn off the hardware or software overtravel checking data bit, using a Control Settings block (Adjust type).
   - Clear the hardware or software overtravel axis fault, using a Reset Fault block.
   - Jog the axis out of the overtravel condition, using a Jog Axis block.
2. When the axis is out of the overtravel range, use the Control Settings block to enable the hardware or software overtravel checking data bit.
3. Re-execute the last command.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Illegal Command With Feedback OFF

This fault (Runtime_fault = 17) indicates that, because feedback to a necessary axis is OFF, the following blocks cannot be executed:

- Analog Offset
- Jog
- Move
- Interpolate Axes

- Gear Axes
- Time Lock Cam
- Position Lock Cam

For example: In a diagram with multitasking, a fault in Task0 might lead to a fault handling routine which turns off feedback, while Task1 continues to execute and commands motion.

To correct the problem:

1. Turn feedback ON for the necessary axis, using a Feedback block.
2. Re-execute the command.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Illegal Command With Feedback ON

This fault (Runtime_fault = 16) indicates that, because feedback to the axis is ON, changes to the following data bits (ordinarily made with a Control Settings block set to Adjust Type) cannot be made:

- Encoder Polarity (B10)
- Servo Output Polarity (B11)

To correct the problem:

1. Turn feedback OFF for the necessary axis, using a Feedback block.
2. Re-execute the Control Settings block.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

RIO Fault Code

RIO_fault_code is an integer value representing the most recent RIO fault condition.

In the Tag Explorer select General System Variables, then select RIO_fault_code in the Tag Window.

Fault Code	Description	Adapter	Scanner
9	Initialization Failure	✔	✔
8	Setup Failure	✔	
7	Setup Failure		✔
6	Failed Getting Inputs	✔	✔
5	Failed Sending Outputs	✔	✔
4	Scanner Failure		✔
3	Failed Getting Data	✔	
2	Failed BTW Request	✔	
1	Failed Sending BTR	✔	
0	No RIO Default	✔	✔

When RIO_fault_code ≥ 4, RIO_status = 1 (OfFLine), the LED on the RIO option is OFF, and Global_fault = 10 (RIO Fault). The different types of RIO faults are explained following.

Initialization Failure

An RIO initialization failure fault (RIO_fault_codem = 9) indicates that the RIO option failed to pass its power-up diagnostics. When RIO_fault_code = 9, RIO_status = 1 (Offline), the LED on the RIO option is OFF, and Global_fault = 10 (RIO Fault).

Setup Failure

An RIO setup failure fault (RIO_fault_code = 7 or 8) indicates that the RIO option could not be set up by the motion controller as configured. When RIO_fault_code = 7 or 8, RIO_status = 1 (Offline), the LED on the RIO option is OFF, and Global_fault = 10 (RIO Fault). Check that the RIO setup data for the adapter or scanner is correct.

Failed Getting Inputs

An RIO failed getting inputs fault (RIO_fault_code = 6) indicates that the motion controller could not successfully read its RIO discrete inputs. When RIO_fault_code = 6, RIO_status = 1 (Offline), the LED on the RIO option is OFF, and Global_fault = 10 (RIO Fault). If you configured the motion controller for adapter mode, check whether the RIO scanner (in the SLC or PLC) is properly scanning the motion controller's I/O. If you configured the RIO option for scanner mode, check whether the discrete inputs actually exist at the configured Scanner I/O Rack Address. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the motion controller's RIO option.

Failed Sending Outputs

An RIO failed sending outputs fault (RIO_fault_code = 5) indicates that the motion controller could not successfully set its RIO discrete outputs. When RIO_fault_code = 5, RIO_status = 1 (Offline), the LED on the RIO option is OFF, and Global_fault = 10 (RIO Fault). If the motion controller is configured for adapter mode, check that the RIO scanner in the SLC or PLC is properly scanning the motion controller's I/O. If the RIO option is configured for scanner mode, check that the discrete outputs actually exist at the configured Scanner I/O Rack Address. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on configuring the motion controller's RIO option.

Scanner Failure

An RIO scanner failure fault (RIO_fault_code = 4) indicates that the motion controller's RIO scanner could not properly establish communication with the configured devices on the RIO link. When RIO_fault_code = 4, RIO_status = 1 (Offline), the LED on the RIO option is OFF, and Global_fault = 10 (RIO Fault). Check that the RIO scanner setup data is correct and that the adapters on the RIO link are powered up and operating properly.

Failed Getting Data

An RIO failed getting data fault (RIO_fault_code = 3) indicates that the motion controller's RIO adapter received a block or discrete transfer from the scanner, but the expected data was not present. Although this fault does not directly cause a global fault, if the condition persists another more serious RIO fault occurs.

Failed BTW Request

An RIO failed BTW request (RIO_fault_code = 2) indicates that the motion controller's RIO adapter could not be enabled to receive a block transfer write from the scanner. This may be caused by improper BT interlocking or logic in the PLC causing block transfers to be sent to the motion controller in the wrong sequence. While this fault does not directly cause a global fault, if the condition persists another more serious RIO fault occurs. See *Using the Remote I/O Adapter* chapter for information on proper block transfer interlocking.

Failed Sending BTR

An RIO failed sending BTR (RIO_fault_code = 1) indicates that the motion controller's RIO adapter could not be enabled to receive a block transfer read from the scanner. This is usually caused by improper BT interlocking or logic in the PLC causing block transfers to be sent to the motion controller in the wrong sequence. While this fault does not directly cause a global fault, if the condition persists another more serious RIO fault undoubtedly occurs. See *Using the Remote I/O Adapter* chapter for information on proper block transfer interlocking.

CNET Fault Code

CNET_fault_code is an integer value representing the most recent CNET fault condition, as shown below. This variable applies only to the 1394 Turbo controllers.

Fault Code	Fault Condition	LED
4	Plug Fault	Solid Red
3	Media Fault	Flashing Red

Fault Code	Fault Condition	LED
2	Incorrect Network Configuration	Flashing Red/ Green
1	Incorrect Node Configuration	Flashing Red
0	No Fault	Solid Green

When CNET_fault_code > 0, CNET_fault = 1, and Global_fault = 16 (if no higher global faults are active).

Note: There are two CNET plug card LEDs – one for channel A and one for channel B. Because these LEDs are internal to the CNET fiber optic plug card, they are not readily visible on this type of plug card.

If a cable is connected to the network access port (NAP) of the CNET plug card, the state of the LEDs is meaningless.

Plug Fault

A CNET Communications fault (CNET_fault_code = 4) indicates that a problem internal to the CNET plug card has caused the card to fail. This fault occurs during CNET initialization.

When CNET_fault_code = 4, Global_fault = 16, CNET_fault = 1, CNET_status = 3 and both CNET plug card LEDs are solid red.

Note: There are two CNET plug card LEDs – one for channel A and one for channel B. Because these LEDs are internal to the CNET fiber optic plug card, they are not readily visible on this type of plug card.

If a cable is connected to the network access port (NAP) of the CNET plug card, the state of the LEDs is meaningless.

To handle or clear this fault:

- Recycle power to the CNET plug card, or

- Execute a Reset CNET Fault block, set to Type Card, as a Direct Command from the Select Direct Command Window on the Online Toolbar.

Executing a Reset CNET Fault block, set to Type Card, will reinitialize the plug card and reset the variables CNET_fault_code and CNET_status to 0.

If executing a Reset CNET Fault block (as described above) fails to clear this fault, the plug card is defective and must be either repaired or replaced.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Media Fault

A CNET Media fault (CNET_fault_code = 3) indicates:

- a wiring problem. Examples of causes of this fault include: an unplugged cable, a broken cable, a cable without a terminator, etc. This fault can occur during either CNET initialization or during diagram execution; or
- no other network nodes are presently configured for ControlNet communications.

When CNET_fault_code = 3, Global_fault = 16, CNET_fault = 1, CNET_status = 3 and the affected CNET plug card LED is flashing red.

If a cable is connected to the network access port (NAP) of the CNET plug card, the state of the LEDs is meaningless.

To clear this fault, either:

- Recycle power to the CNET plug card, or

- Find and fix the problem (either fix the wiring problem, or configure other network nodes for ControlNet communications). Then, execute a Reset CNET Fault block, set to type All Faults, as a Direct Command from the Select Direct Command Window on the Online Toolbar. Executing a Reset CNET Fault block, set to Type All Faults, will reset the variables CNET_fault_code and CNET_status to 0.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Incorrect Network Configuration

A CNET Incorrect Network Configuration fault (CNET_fault_code = 2) indicates problems with the network configuration. For example, if the CNET network is configured for redundant media, a channel A cable connected to a channel B port (or a channel B cable connected to a channel A port) would cause this fault.

When CNET_fault_code = 2, Global_fault = 16, CNET_fault = 1, CNET_status = 3 and the affected CNET plug card LED is flashing red and green.

If a cable is connected to the network access port (NAP) of the CNET plug card, the state of the LEDs is meaningless.

To handle or clear this fault:

- Find and fix the network configuration problem. Check that all channel A cable connections are in fact connected only to channel A ports on all network devices, and that all channel B cable connections are in fact connected only to channel B ports.

Then either:

- Recycle power to the CNET plug card, or
- Execute a Reset CNET Fault block, set to type All Faults, as a Direct Command from the Select Direct Command Window on the Online Toolbar. Executing a Reset CNET Fault block, set to Type All Faults, will reset the variables CNET_fault_code and CNET_status to 0.

If recycling power or executing a Reset CNET Fault block (as described above) fails to clear this fault, the plug card is defective and must be repaired or replaced.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

Incorrect Node Configuration

A CNET Incorrect Node Configuration fault (CNET_fault_code = 1) occurs if multiple nodes on the ControlNet network are set to the same Media Access Control ID (in the CNET page of the Configure Control Options dialog box). This fault occurs during CNET initialization.

When CNET_fault_code = 1, Global_fault = 16, CNET_fault = 1, CNET_status = 3 and the affected CNET plug card LED is flashing red.

If a cable is connected to the network access port (NAP) of the CNET plug card, the state of the LEDs is meaningless.

To clear this fault, either:

- Recycle power to the CNET plug card, or
- Fix the MAC ID configuration problem. Then execute a Reset CNET Fault block, set to Type All Faults, as a Direct Command from the Select Direct Command Window on the Online Toolbar. Executing a Reset CNET Fault block, set to Type All Faults, will reset the variables CNET_fault_code and CNET_status to 0.

This fault condition does not appear as an independent fault variable in either the Tag Window or the Expression Builder.

DH-485 Fault Code

DH485_fault_code is an integer value representing the fault status of the most recent DH-485 transaction initiated by the motion controller, as shown in the table below.

In the Tag Explorer select General System Variables, then select DH-485_fault_code in the Tag Window.

Fault Code	Description
5	Remote Device Response STS Byte
4	Command Failed
3	Response Timeout
2	Transaction ID Mismatch
1	Bad Command
0	No DH-485 Fault

When DH485_fault_code > 0, DH485_general_fault = 1 (DH-485 fault). When DH485_fault_code = 0, the remote device received and accepted (acknowledged) the message sent by the motion controller.

When DH485_fault_code ≥ 5, the remote device received the message from the motion controller but may not have executed the requested action. In this case, DH485_fault_code is the value of the status (STS) byte of the response from the remote device as received by the motion controller. Consult the documentation for the remote device to determine the meaning of DH-485 status codes greater than or equal to 5.

Important: Although DH485_fault_code is a decimal value in GML Commander, the STS byte is most often documented elsewhere as a hexadecimal value.

The four lowest-numbered DH-485 faults are described below.

Command Failed

A command failed fault (DH485_fault_code = 4) indicates that the device to which the message was sent did not accept (negative acknowledged) the message. Verify that the remote DH-485 device is configured to accept DH-485 messages from the motion controller.

Response Timeout

A response timeout fault (DH485_fault_code = 3) indicates that a message was successfully sent to the remote device, but no response was received from the remote device within the timeout period. The timeout period is approximately one second per valid network node address. For example, if all 31 DH-485 nodes are used, the timeout period is about 31 seconds. Verify that the remote device is operating properly.

Transaction ID Mismatch

A transaction ID mismatch fault (DH485_fault_code = 2) indicates that a message was successfully sent to the remote device, but that the transaction ID of the received response was not the same as that of the sent message. This error occurs when messages and responses get out of sequence on the DH-485 network (the motion controller received the response from another device's message). This condition is most often due to heavy network traffic combined with a slow response remote device.

Bad Command

A bad command fault (DH485_fault_code = 1) indicates that the message was not sent to the remote device. This fault is most often caused by improper message syntax, sending a message with the motion controller disconnected from the DH-485 network, or attempting to send a message before the previous one has completed or timed out.

DSP Feedback Fault 1394

A DSP feedback fault occurs when the 1394 motion controller fails to communicate to the DSP subsystem, which interfaces with the 1394 axis modules. This fault indicates a hardware problem with the control board in the system module.

After the underlying problem has been detected and remedied, clear this fault with a Reset Fault block set to Reset 1394, or by pressing the controller Reset button.

In the Tag Explorer select General System Variables, then select DSP_feedback_fault_1394 in the Tag Window.

SLC Fault Code

When a fault occurs within the SLC interface, the SLC_fault_code variable determines the type of fault, as follows:

Fault Code	Type	Description
0	No fault	When no SLC faults have occurred, the variables SLC_fault and SLC_fault_code both = 0
1	SLC process fault	Occurs when the dedicated SLC process fault bit is set to 1 by the SLC application program. This allows the SLC application program to generate a fault condition in the 1394 Turbo.
2	Power failure in an SLC rack	Occurs when one of the external SLC racks has a power failure. Unlike the other faults, this fault is automatically cleared when SLC rack power is restored
3	SLC interface hardware fault	Occurs when the SLC interface circuitry has failed power-up diagnostics and is non-recoverable. If this fault persists after power-cycle, the 1394 Turbo must be repaired.

These faults apply to the 1394 Turbo, and do not apply to the SLC 500 CPU.

Resolver Loss Fault 1394

A resolver loss fault occurs when the 1394 motion controller detects that one or more of the resolver signals, associated with the specified axis, is missing.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Resolver_loss_fault_1394 in the Tag Window.

Drive Hard Fault 1394

A drive hard fault occurs when the system module, or any axis module, detects a fault condition. When this condition occurs, the status LED on the front cover of the 1394 system module flashes red. You can identify the offending module(s) by checking the status of the 1394 axis module hard fault variables, and the 1394 system module hard fault variable.

ATTENTION: The hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

When this fault occurs:

1. Determine whether the fault is an axis or system fault, and locate the specific underlying fault.
2. Fix the underlying cause of the fault.
3. To clear this fault:
   - Use a Reset Fault block, set to Reset 1394,or
   - Press the reset switch on the controller, or
   - Cycle control power.

In the Tag Explorer select General System Variables, then select Drive_hard_fault_1394 in the Tag Window.

Axis Module Hard Fault 1394

An axis module hard fault indicates that the associated 1394 axis module has detected a fault condition. You can determine the specific fault condition by checking the various axis fault variables.

Axis_module_hard_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the associated 1394 axis module has detected a fault condition (listed below), or

- 0 (false) if not.

When Axis_module_hard_fault_1394 = 1, Drive_hard_fault_1394 = 1.

The occurrence of one of the following underlying faults triggers the Axis_module_hard_fault_1394:

- Axis_bus_loss_fault_1394
- Axis_Drive_over_temp_fault_1394
- Axis_it_fault_1394
- Axis_power_fault_1394

To clear this fault, you must clear the underlying fault with a Reset Fault block, set to Reset 1394.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_module_hard_fault_1394 in the Tag Window.

System Module Hard Fault 1394

System_module_hard_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the 1394 system module has detected a fault condition, and
- 0 (false) if not.

When System_module_hard_fault_1394 = 1, Drive_hard_fault_1394 = 1.

The occurrence of one of the following underlying faults triggers the System_module_hard_fault_1394:

- System_bus_over_voltage_flt_1394
- System_bus_under_voltage_flt_1394
- System_control_power_fault_1394
- System_ground_fault_1394

- System_over_temp_fault_1394
- System_phase_loss_fault_1394
- System_smrt_pwr_i_limit_fault_1394
- System_smrt_pwr_pre-charge_fault_1394
- System_smrt_pwr_shunt_timeout_fault_1394

To clear this fault, you must clear the underlying fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear the underlying fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before program execution can be resumed.

In the Tag Explorer select General System Variables, then select System_module_hard_fault_1394 in the Tag Window.

Axis Power Fault 1394

Axis_power_fault_1394 is a logical (Boolean) variable with a value of:

- 1 (true) if an enabled 1394 axis module detects any of the following conditions:
  - Instantaneous 300% over-current condition
  - Power block short circuit
  - Power block over-temperature
  - Loss of base drive to power block
- 0 (false) if not.

When Axis_power_fault_1394 = 1, Axis_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_power_fault_1394 in the Tag Window.

Axis Bus Loss Fault 1394

Axis_bus_loss_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if an enabled 1394 axis module detects that the bus voltage level has fallen below acceptable limits, and
- 0 (false) if not.

When Axis_bus_loss_fault_1394 = 1,

Axis_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_bus_loss_fault_1394 in the Tag Window.

Axis Motor Over Temp Fault 1394

An axis motor over temp fault occurs when the 1394 controller detects that the thermal fault switch input to the control has opened. This condition can occur when the motor gets too hot.

Axis_Motor_over_temp_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the 1394 controller detects that the thermal fault switch input to the control has opened (i.e., the motor gets too hot), and
- 0 (false) if not.

When Axis_Motor_over_temp_fault_1394 = 1, Global Fault = 5.

Clearing this fault clears Global Fault.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with Clear Axis Fault, set to clear Motor Thermal Fault.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer select both Axis System Variables and the desired physical axis, then select Axis_Motor_over_temp_fault_1394 in the Tag Window.

Axis Drive Over Temp Fault 1394

An axis drive over temp fault occurs when an enabled 1394 axis module detects that the ambient air temperature, inside the associated axis module, becomes too hot.

Axis_Drive_over_temp_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if an enabled 1394 axis module detects that the ambient air temperature, inside the associated axis module, becomes too hot, or
- 0 (false) if not.

This fault could occur for any of the following reasons:

- Excessive ambient temperature.
- Insufficient air flow.
- Excessive acceleration and deceleration duty cycle.

When Axis_Drive_over_temp_fault_1394 = 1, Axis_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_Drive_over_temp_fault_1394 in the Tag Window.

Axis It Fault 1394

An axis it fault occurs when an enabled 1394 axis module detects that the motor current integrated over time (It) exceeds the capability of the power block.

Axis_it_fault_1394 is a logical (Boolean) variable, with a value or:

- 1 (true) if an enabled 1394 axis module detects that the motor current integrated over time (It) exceeds the capability of the power block, and
- 0 (false) if not.

When Axis_it_fault_1394 = 1, Axis_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_it_fault_1394 in the Tag Window.

System Bus Over Voltage Fault 1394

The System_bus_over_voltage_flt activates if drive bus reaches 810V. This fault can be caused by exceeding the shunt regulator duty cycle.

System_bus_over_voltage_flt_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the drive bus reaches 810V, and
- 0 (false) if not.

This fault can be caused by exceeding the shunt regulator duty cycle.

When System_bus_over_voltage_flt_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer select General System Variables, then select System_bus_over_voltage_flt_1394 in the Tag Window.

System Bus Under Voltage Fault 1394

If at any time the drive bus falls below 275V, the sysem_bus_undr_voltage_flt_1394 activates.

System_bus_undr_undr_voltage_flt_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the drive bus falls below 275V at any time, and
- 0 (false) if not.

When System_bus_under_voltage_flt_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before the System Bus can be reapplied.

In the Tag Explorer select General System Variables, then select System_bus_under_voltage_flt_1394 in the Tag Window.

System Over Temp Fault 1394

The System_over_temp_fault_1394 is activated if the ambient air temperature inside the system module becomes too hot.

System_over_temp_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the ambient air temperature inside the system module becomes too hot, and
- 0 (false) if not.

This fault could occur for any of the following reasons:

- Excessive ambient temperature,
- Insufficient air flow, or
- Excessive acceleration and deceleration duty cycle.

When System_over_temp_fault_1394 = 1,
System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before program execution can be resumed.

In the Tag Explorer select General System Variables, then select System_over_temp_fault_1394 in the Tag Window.

System Control Power Fault 1394

The System_control_power_fault_1394 occurs when the 1394 system module detects that voltages, generated by the power supply for the control, are below acceptable limits.

System_control_power_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the 1394 system module detects that voltages, generated by the power supply for the control, are below acceptable limits, and
- 0 (false) if not.

When System_control_power_fault_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before program execution can be resumed.

In the Tag Explorer select General System Variables, then select System_control_power_fault_1394 in the Tag Window.

System Phase Loss Fault 1394

A System_phase_loss_fault_1394 occurs when the 1394 system module detects that one or more phases of the main power supply is missing.

System_phase_loss_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the 1394 system module detects that one or more phases of the main power supply are missing, and
- 0 (false) if not.

When System_phase_loss_fault_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before program execution can be resumed.

In the Tag Explorer select General System Variables, then select System_phase_loss_fault_1394 in the Tag Window.

System Ground Fault 1394

A System_ground_fault_1394 occurs when the 1394 system module detects that there is significant current flowing through the earth ground connection to the drive. This usually results from improper installation of the 1394 drive system.

System_ground_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (true) if the 1394 system module detects that there is significant current flowing through the earth ground connection to the drive, and
- 0 (false) if not.

This fault usually results from improper installation of the 1394 drive system.

When System_ground_fault_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

Clear this fault with a Reset Fault block, set to Reset 1394.

Important: Do not use an Equation block to clear this fault. The Reset Fault block not only resets this variable to zero, it also resets the Drive OK Relay inside the 1394 system module. This relay must be reset before program execution can be resumed.

In the Tag Explorer select General System Variables, then select System_ground_fault_1394 in the Tag Window.

System Smart Power I Limit Fault 1394

The Programmable Integrated Controller (PIC) processor, in the Smart Power system module, monitors the DC link current to protect the DC bus pins when maximum power is being consumed in either normal operation or a common bus mode. System_smrt_pwr_i_limit_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (True) if either the 64 amperes instantaneous current limit or the 34 amperes Root Mean Square (RMS) continuous current limit is exceeded by 5%, or
- 0 (False) if not.

When System_smrt_pwr_i_limit_fault_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

When System_smrt_pwr_i_limit_fault_1394 = 1, System_module_fault_status_1394 = 9.

System Smart Power Pre-Charge Fault 1394

System_smrt_pwr_pre-charge_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (True) if the Smart Power system module DC bus fails to reach 458 Volts after 2 or 3 phases are present, or
- 0 (False) if not.

When System_smrt_pwr_pre-charge_fault_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

When System_smrt_pwr_pre-charge_fault_1394 = 1, System_module_fault_status_1394 = 10.

System Smart Power Shunt Timeout Fault 1394

System_smrt_pwr_shunt_timeout_fault_1394 is a logical (Boolean) variable, with a value of:

- 1 (True) if the 34 Amperes Root Mean Square (RMS) continuous or 200 Amperes peak rating of the Smart Power system module shunt resistor is exceeded by 5%, or
- 0 (False) if not.

When System_smrt_pwr_shunt_timeout_fault_1394 = 1, System_module_hard_fault_1394 = 1, and Drive_hard_fault_1394 = 1.

Clearing this fault clears Drive_hard_fault_1394.

ATTENTION: A hard fault is a serious condition! Be certain to correct the underlying cause of this fault, before clearing the fault and reactivating the drive.

When System_smrt_pwr_shunt_timeout_fault_1394 = 1, System_module_fault_status_1394 = 8.

Status Variables

Status variables allow any of the various status conditions within the motion controller to be used as elements in an expression. Mathematical variables for axis status, program status, and the front panel status LEDs, as well as logical variables for individual axis status conditions, are provided as shown in the table below.

Variable	Units	Servo	Master Only	Virtual	Imaginary
Axis_status	Integer (0 – 14)	✓	✓	✓	✓
Accel_status	0 (False) or 1 (True)	✓			✓
Decel_status	0 (False) or 1 (True)	✓			✓
Encoder_Filter_Lag_Saturation_Status	0 (False) or 1 (True)		✓	✓	
Feedback_status	0 (False) or 1 (True)	✓			✓
Homing_status	0 (False) or 1 (True)	✓	✓	✓	
Jog_status	0 (False) or 1 (True)	✓			✓
Lock_status	0 (False) or 1 (True)	✓			
Move_status	0 (False) or 1 (True)	✓			✓
Gearing_status	0 (False) or 1 (True)	✓			✓
Registration_status	0 (False) or 1 (True)	✓	✓	✓	
TCAM_status	0 (False) or 1 (True)	✓			✓
PCAM_status	0 (False) or 1 (True)	✓			✓
PCAM_profile_status	0 (False) or 1 (True)	✓			✓
PCAM_pending_pr_status	0 (False) or 1 (True)	✓			✓
PCAM_auto_corr_status	0 (False) or 1 (True)	✓			✓

Variable	Units	Servo	Master Only	Virtual	Imaginary
Watch_Pos_status	0 (False) or 1 (True)	✓	✓	✓	✓
Output_limit_status	0 (False) or 1 (True)	✓			
AxisLink_status	0 (False) or 1 (True)			✓	
Interp0_status	0 (False) or 1 (True)				
Interp1_status	0 (False) or 1 (True)				
Accel_status_Interp0	0 (False) or 1 (True)				
Accel_status_Interp1	0 (False) or 1 (True)				
Decel_status_Interp0	0 (False) or 1 (True)				
Decel_status_Interp1	0 (False) or 1 (True)				
Merge_status_Interp0	0 (False) or 1 (True)				
Merge_status_Interp1	0 (False) or 1 (True)				
Status_LEDs	Integer from 0 to 7				
Program_status	Integer from 0 to 2				
RIO_status	Integer from 0 to 5				
CNET_status	Integer from 0 to 3				
SLC_status	Integer from 0 to 3				
DH485_status	Integer from 0 to 2				
Axis_it_limit_status_1394	0 (False) or 1 (True)				
Axis_i_limit_status_1394	0 (False) or 1 (True)				
System_module_fault_status_1394	Integer from 0 to 10				
Axis_module_fault_status_1394	Integer from 0 to 11				
System_smrt_pwr_i_limit_status_1394	0 (False) or 1 (True)				
System_smrt_pwr_shunt_it_disable_1394	0 (False) or 1 (True)				
System_smrt_pwr_shunt_timeout_status_1394	0 (False) or 1 (True)				

All status variables—except Status_LEDs—are read-only.

Status_LEDs can also be used within an expression, and can be used to display a code on the motion controller's front panel Status LEDs.

Each of the variables in the table above is described in the following pages.

Axis Status

Axis_status is an integer value representing the present status of a specific axis.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_status in the Tag Window.

The present status of the specified axis appears as a value, according to the following table. The table shows the corresponding message that appears in an axis status field in the runtime display (if enabled).

Status Code	Description	Runtime Display	Servo	Master Only	Virtual	Imaginary
14	AxisLink timeout	AXL FLT			✓	
13	AxisLinkFailed (axis not found)	AXL FLT			✓	
12	Virtual Axis Not Enabled	AXS OFF			✓	
11	Drive Fault	DRV FLT	✓			
10	Position Error Tolerance Fault	ERR FLT	✓			
9	Hardware Overtravel Fault	HRD LIM	✓			
8	Software Travel Limits Fault	SFT LIM	✓			
7	Encoder Noise or Loss Fault	ENC FLT	✓	✓		
6	Feedback OFF or Virtual Axis Enabled	SRV OFF SRV OFF	✓	✓	✓	
5	Servo Output Limited	OUT LIM	✓			

Status Code	Description	Runtime Display	Servo	Master Only	Virtual	Imaginary
4	Homing	HOMING	✓	✓	✓	
3	Moving or Executing Time-Lock Cam	MOVING	✓			✓
2	Jogging	JOGGING	✓			✓
1	Axis Unlocked	UNLOCK	✓			✓
0	Axis Locked	LOCKED	✓			

Axis status conditions are prioritized from highest to lowest in the order shown in the table. The higher the status value, the higher its priority (or severity). This means that when a given status is active, other status conditions of lower priority may also be active. For instance, if a hardware overtravel fault is active for an axis, a servo output limited or feedback OFF condition may also be active on the axis. When the highest priority status is no longer active, Axis_status takes the value of the next highest priority condition.

Seven of the Axis_status variable values (7 - 11, 13, 14) are fault conditions. They are identical to the top seven Axis_fault variable values as shown in the table below.

Axis Status	Axis Fault	Description	Runtime Display
14	7	AxisLink timeout	AXL FLT
13	6	AxisLinkFailed (axis not found)	AXL FLT
11	5	Drive Fault	DRV OFF
10	4	Position Error Tolerance Fault	ERR FLT
9	3	Hardware Overtravel Fault	HRD FLT
8	2	Software Travel Limits Fault	SFT LIM
7	1	Encoder Noise or Loss Fault	ERR FLT

See the individual axis status conditions for an explanation of each axis status condition.

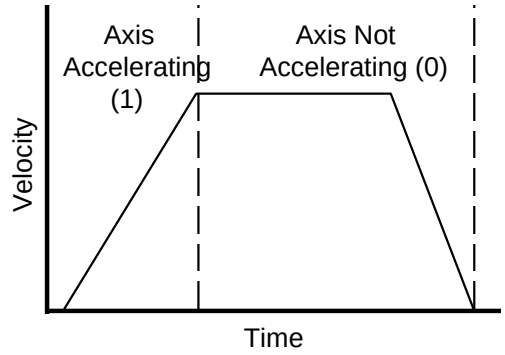
Acceleration Status

Accel_status is a logical (Boolean) variable which has values of:

- 1 (true) if the axis is accelerating
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select Accel_status in the Tag Window.

The value of the Accel_status variable indicates whether or not the axis is currently accelerating because of a previous Interpolate Axes, Move Axis or Jog Axis block.



Note: The acceleration, caused by electronic gearing, time-lock cams, or position-lock cams, does not affect the acceleration status value.

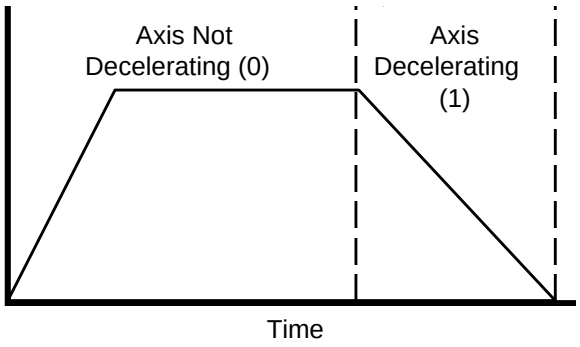
Deceleration Status

Decel_status is a logical (Boolean) variable which has values of:

- 1 (true) if the axis is decelerating, and
- 0 (false) if not.

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select Decel_status in the Tag Window.

The value of the Decel_status variable indicates whether or not the axis is currently decelerating because of a previous Interpolate Axes, Move Axis, Jog Axis, or Stop Motion block.



Deceleration caused by electronic gearing, time-lock cams, or position-lock cams does not affect the deceleration status value.

Encoder Filter Lag Saturation Status

Encoder_Filter_Lag_Saturation_Status is a logical (Boolean) variable, which has a value of

- 1 (true) if the Lag Limit, set in the Feedback page of the Configure Axis Use dialog box, has been exceeded, and
- 0 (false) if not.

Note: This variable appears in the Tag Window only if Master Only Axis Type is selected in the General Page of the Configure Axis Use dialog box, and Encoder Filter is selected in the Feedback Page of that dialog box.

Feedback Status

Feedback_status is a logical (Boolean) variable which has values of:

- 1 (true) if the feedback loop of a servo axis is enabled
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Feedback_status in the Tag Window.

You can directly enable feedback using a Feedback block with Feedback On selected. You can also enable feedback using a Home Axis block, depending on the selected homing procedure.

You can directly disable feedback using a Feedback block with Feedback Off selected. Feedback can also be disabled by a Stop Motion block (depending on the selected mode), and by certain axis faults, depending on the fault action setups.

When Feedback_status = 1, Axis_fault = 0 and Axis_status ≤ 5 if no faults are active on the axis. If no other faults are active on any axis, Global_fault = 0.

When Feedback_status = 0, Axis_fault = 0 and Axis_status = 6 if no faults are active on the axis. If no faults are active on any axis, Global_fault = 0.

Homing Status

Homing_status is a logical (Boolean) variable which has values of:

- 1 (true) if the axis is homing
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or virtual axis, then select Homing_status in the Tag Window.

When Homing_status = 1, Axis_status = 4 if no faults are active on the axis.

Jog Status

Jog_status is a logical (Boolean) variable which has values of

- 1 (true) if the axis is jogging, and
- 0 (false) if not.

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select Jog_status in the Tag Window.

Jogging is initiated by a Jog Axis block, and stopped by a Stop Motion or Change Dynamics block.

When Jog_status = 1, Axis_status = 2 if no faults are active on the axis.

Lock Status

Lock_status is a logical (Boolean) variable which has values of:

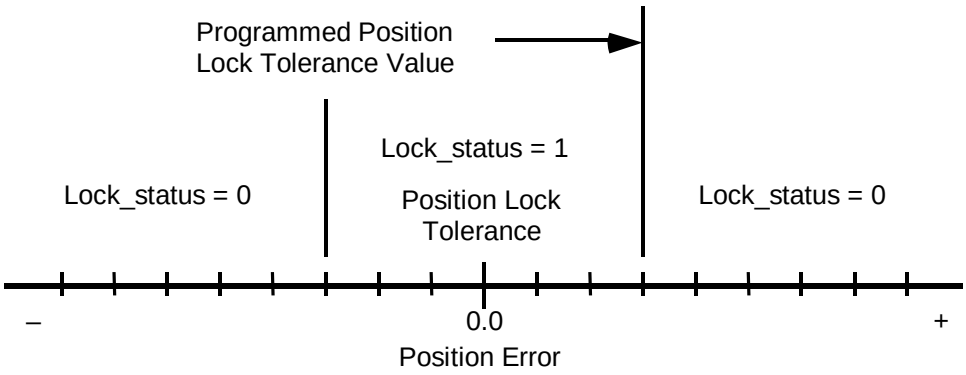
- 1 (true) if the axis is locked onto its command position
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Lock_status in the Tag Window.

The value of the Lock_status variable indicates whether the axis is currently locked. An axis is locked whenever either of the following two sets of conditions are true:

- Electronic gearing OFF and no position-lock cam or interpolated motion in progress:
  No motion (move, jog, or time-lock cam) in progress
  Position error ≤ position lock tolerance
- Electronic gearing ON or position-lock cam or interpolated motion in progress:
  Position error ≤ position lock tolerance

The position Lock Tolerance is set in the Positioning page of the selected axis' Configure Axis Use dialog box, and specifies how much position error the motion controller tolerates in a locked condition. As such, it is one of the factors that determines positioning accuracy. The Position Lock Tolerance value is interpreted as a ± quantity.



See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on position lock tolerance.

When Lock_status = 1, Axis_status = 0 if no faults are active and no motion is being commanded on the axis.

When Lock_status = 0, Axis_status = 1 if no faults are active and no motion is being commanded on the axis.

Move Status

Move_status is a logical (Boolean) variable which has values of:

- 1 (true) if the axis is being commanded to move
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select Move_status in the Tag Window.

Moves are initiated by a Move Axis block and stopped by a Stop Motion block.

When Move_status = 1, Lock_status = 0, and Axis_status = 3 if no faults are active on the axis.

Gearing Status

Gearing_status is a logical (Boolean) variable which has values of:

- 1 (true) if electronic gearing is active for the axis
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select Gearing_status in the Tag Window.

Gearing is activated by a Gear Axes block and stopped by a Disable Gearing or Stop Motion block.

When Gearing_status = 1, Axis_status ≤ 5 if no faults are active on the axis.

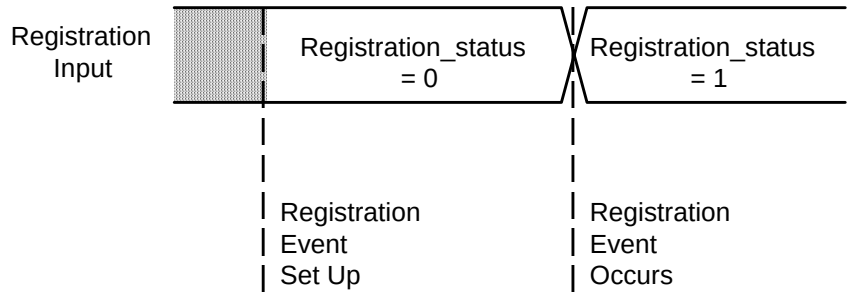
Registration Status

Registration_status is a logical (Boolean) variable which has values of:

- 0 (false) after a registration event has been set up and before the registration event occurs
- 1 (true) after the registration event occurs

In the Tag Explorer, select both Axis System Variables and the desired physical or virtual axis, then select Registration_status in the Tag Window.

The value of the Registration_status variable indicates whether a previously set up registration event for the axis has occurred.



Important: Until the registration event is set up, the value of the Registration_status variable is undefined and may be 0 or 1.

If no registration event has previously been set up for the axis, Registration_status is 0. If a previous registration event has been set up, Registration_status is 1, if the previous event occurred and 0 if not.

Registration events are set up using a Watch Control block with Arm and Registration selected, and canceled using the Watch Control block with Disarm and Registration selected.

Time Lock Cam Status

TCAM_status is a logical (Boolean) variable which has values of:

- 1 (true) if the axis is executing a time-lock cam
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select TCAM_status in the Tag Window.

Time-lock cams are initiated by a Time Lock Cam block, and can be stopped by a Stop Motion block.

When TCAM_status = 1, Lock_status = 0, and Axis_status = 3 if no faults are active on the axis.

Position Lock Cam Status

PCAM_status is a logical (Boolean) variable which has values of:

- 1 (true) if a position-lock cam has been initiated and is active for the axis
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select PCAM_status in the Tag Window.

Position-lock cams are initiated using a Position Lock Cam block, and stopped using a Disable Position Lock CAM or Stop Motion block, depending on the selected mode.

When PCAM_status = 1, Axis_status ≤ 5 if no faults are active on the axis.

Position Lock Cam Profiling Status

PCAM_profile_status is a logical (Boolean) variable which has values of:

- 1 (true) if a position-lock cam has been initiated for the axis and the specified master axis is within the range defined by the master cam table
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select PCAM_profile_status in the Tag Window.

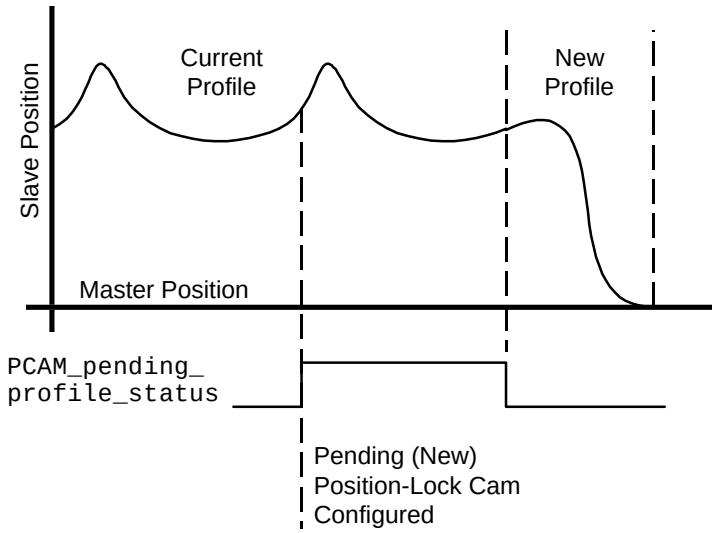
Position-lock cams are initiated by a Position Lock Cam block and stopped by a Disable PCAM or Stop Motion block, depending on the selected mode.

When PCAM_profile_status = 1, PCAM_status = 1, and Axis_status ≤ 5 if no faults are active on the axis.

Position Lock Cam Pending Profile Status

PCAM_pending_profile_status is a logical (Boolean) variable which has values of:

- 1 (true) if a new position-lock cam profile is currently pending for the axis
- 0 (false) if not



In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select PCAM_pending_profile_status in the Tag Window.

Pending position-lock cams are set up by choosing Pending Position-Lock from the Cam Type pop-up menu in the Configure Cam block.

Position Lock Cam Auto-Correction Status

PCAM_auto_correction_status is a logical (Boolean) variable which has values of:

- 1 (true) if auto-correction is currently enabled on the specified axis
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select PCAM_auto_correction_status in the Tag Window.

Auto-correction is enabled and disabled in the Configure Cam block. Auto-correction on an axis is automatically disabled when the position-lock cam, of which the axis is a slave, is stopped.

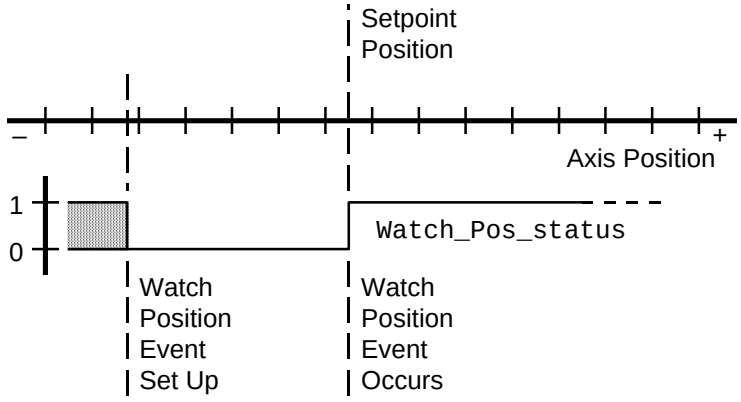
Watch Position Status

Watch_Pos_status is a logical (Boolean) variable which has values of:

- 0 (false) after a watch position event has been set up and before the watch position event occurs
- 1 (true) after the watch position event occurs

In the Tag Explorer, select both Axis System Variables and the desired physical or imaginary axis, then select Watch_Pos_status in the Tag Window.

The Watch_Pos_status variable indicates whether the watch position event—previously set up for the axis—has occurred.



Important: Until the watch position event is set up, the Watch_Pos_status variable value is undefined.

If no watch position event has previously been set up for the axis, Watch_Pos_status is 0.

If a previous watch position event has been set up, Watch_Pos_status is:

- 1 if the previous event occurred
- 0 if not

Set up Watch Position events in a Watch Control block, with Arm and Watch Position. Watch Position events are canceled when the watch position event occurs, or by using a Watch Control block with Disarm selected.

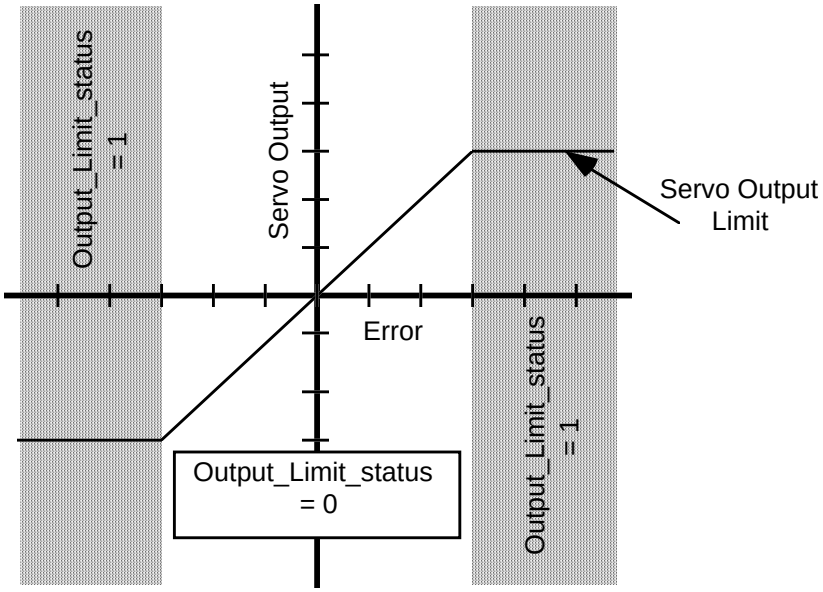
Output Limit Status

Output_limit_status is a logical (Boolean) variable which has values of:

- 1 (true) if the servo output voltage of the axis is being limited
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Output_limit_status in the Tag Window.

The value of the Output_limit_status variable indicates whether the output voltage of the axis is being limited to the previously entered servo output limit value.



The servo output limit is set in the motion controller's machine setup menu, or by using a Motion Settings block with Set Output Limit selected. The servo output limit specifies the maximum voltage (±) that the servo output is allowed to produce. See the *Setup* section of the *Installation and Setup* manual for your motion controller for more information on the servo output limit.

When Output_limit_status = 1, Axis_status = 5 if no faults are active. When Output_limit_status = 0, Axis_status ≤ 4 if no faults are active on the axis.

AxisLink Status

AxisLink_status is a logical (Boolean) variable which has values of:

- 1 (true) if the specified AxisLink virtual axis is enabled
- 0 (false) if not

In the Tag Explorer, select both Axis System Variables and the desired virtual axis, then select Output_limit_status in the Tag Window.

Virtual axes are enabled and disabled by a Virtual Axis Control block.

When AxisLink_status = 1, Axis_fault = 0 and Axis_status = 6 if no faults are active on the axis, and Global_fault = 0 if no other faults are active on any axis.

When AxisLink_status = 0, Axis_status = 12 if no faults are active on the axis, and Global_fault = 0 if no faults are active on any axis.

Interpolator Status

Interp0_status and Interp1_status are logical (Boolean) variables that have values of:

- 1 (true) if the corresponding interpolator is commanding motion
- 0 (false) if not

In the Tag Explorer select General System Variables, then select either Interp0_status or Interp1_status in the Tag Window.

While Interp0_status or Interp1_status = 1 (true), Axis_status ≤ 5 for each axis commanded by the interpolator if no faults are active on the axis. S

Interpolator Acceleration Status

Accel_status_Interp0 and Accel_status_Interp1 are logical (Boolean) variables that have values of:

- 1 (true) if the corresponding interpolator is commanding vector acceleration of the associated axes, and
- 0 (false) if not.

In the Tag Explorer select General System Variables, then select either Accel_status Interp0 or Accel_status_Interp1 in the Tag Window.

Interpolator Deceleration Status

Decel_status_Interp0 and Decel_status_Interp1 are logical (Boolean) variables which have values of:

- 1 (true) if the corresponding interpolator is commanding vector deceleration of the associated axes
- 0 (false) if not

In the Tag Explorer select General System Variables, then select either Decel_status Interp0 or Decel_status_Interp1 in the Tag Window.

Interpolator Merging Status

Merge_status_Interp0 and Merge_status_Interp1 are logical (Boolean) variables which have values of:

- 1 (true) if the corresponding interpolator has been commanded to merge the next move with the currently executing move
- 0 (false) if not

In the Tag Explorer select General System Variables, then select either Merge_status Interp0 or Merge_status_Interp1 in the Tag Window.

System Bus Up 1394

System_bus_up_1394 is a logical (Boolean) variable with a value of:

- 1 (true) if the 1394 system module is up (i.e., running without both voltage loss and phase loss), and
- 0 (false) if not.

When System_bus_up_1394 = 0, System_bus_under_voltage_fault_1394 = 1 and System_phase_loss_fault_1394 = 1.

Status LEDs

Three general purpose status LEDs are provided on the front panel of the motion controller. Labeled STATUS 0, STATUS 1, and STATUS 2, these LEDs indicate the results of the power-up diagnostics, performed whenever power is applied. After the power-up diagnostics, the three status LEDs can be used for any desired purpose.

Status_LEDs is a mathematical variable used to turn the status LEDs ON and OFF. By using an Equation block to assign an integer value from 0 to 7 to the Status_LEDs variable, you can turn ON or OFF each of the LEDs in combination, as follows:

Status_LEDs Value	Status0	Status1	Status2
0	OFF	OFF	OFF
1	ON	OFF	OFF
2	OFF	ON	OFF
3	ON	ON	OFF
4	OFF	OFFF	ON
5	ON	OFF	ON
6	OFF	ON	ON
7	ON	ON	ON

You can also use the Status_LEDs variable in an expression to determine the current state of the three status LEDs. This lets you make decisions based on the state of the status LEDs.

In the Tag Explorer select General System Variables, then select LEDs_status in the Tag Window.

Program Status

Program_status is an integer value representing the current state of the application program in the motion controller. When used in an expression in a GML Commander diagram, the Program_status variable always has values of 1 indicating that the application program is running.

Ordinarily, you will use the Show Program Status block in the Online Toolbar's Select Direct Command window to check the Program_status variable, rather than use this variable directly in an expression.

In the Tag Explorer select General System Variables, then select Program_status in the Tag Window.

RIO Status

RIO_status is an integer value representing the present status of Remote I/O communications. The color of the diagnostic LED, for the appropriate channel on the front panel of the RIO option, also indicates RIO status.

In the Tag Explorer select General System Variables, then select RIO_status in the Tag Window.

Value	Description	LED Color
5	Program Mode	No Change
4	Non-Recoverable Fault	Red
3	Recoverable Error (Failing)	Flashing Red
2	Standby	Flashing Green
1	Offline	Off
0	Online	Green

CNET Status

CNET_status is an integer value representing the present status of CNET communications, as shown below.

Value	Description
3	CNET Fault
2	Failing
1	Offline
0	Online

CNET Fault status (CNET_status = 3) indicates that a CNET fault has occurred (i.e., CNET_fault = 1). For information on the cause of the particular fault, check the value of the CNET_fault_code variable, then refer to the chapter on *Fault Variables* in this manual for that CNET_fault_code value and description.

Failing status (CNET_status = 2) indicates that the CNET plug card is experiencing temporary errors.

Do not attempt to undertake any corrective steps at this time.

The CNET plug card attempts to automatically correct these errors. If the plug card is able to correct these temporary errors, CNET operation returns to Online status (CNET_status = 0). If not, a CNET fault (CNET_status = 3) occurs.

Offline status (CNET_status = 1) can indicate a number of different conditions, such as:

- No power.
- The CNET plug card is conducting a self test.
- The local node is not configured to go online.

Online status (CNET_status = 0) indicates normal CNET operation.

SLC Status

SLC_status is an integer value identifying the mode in which the SLC is operating, as follows:

Status	Description
0	Program Mode
1	Run Mode
2	(not used)
3	Faulted Mode

When SLC_status = 0 (Program mode), the SLC is in the mode for creating or programming a ladder logic program in the SLC.

When SLC_status = 1 (Run mode), the SLC is executing or running a previously programmed SLC ladder logic program.

When SLC_status = 3 (Faulted mode), a system-level fault has occurred in the SLC. Refer to your SLC documentation for a description of these faults, and how to handle them.

In the Tag Explorer select General System Variables, then select SLC_status in the Tag Window.

SLC_M0_Update_Acknowledge

SLC_M0_Update_Acknowledge is a logical (Boolean) variable with a value of:

- 1 (true) if the 1394 Turbo controller affirmatively acknowledges an SLC_M0_Update_Request from the SLC, and permits the MO file transfer, and
- 0 (false) if the 1394 Turbo controller has not acknowledged the SLC_M0_Update_Request (or if the transfer is complete).

SLC M0 Update Request

SLC_M0_Update_Request is a logical (Boolean) variable with a value of:

- 1 (true) if the SLC is requesting to send M0 file data to the 1394 Turbo controller, and
- 0 if not.

An SLC_M0_Update_Request precedes an SLC_M0_Update_Acknowledge variable.

SLC M1 Update Acknowledge

SLC_M1_Update_Acknowledge is a logical (Boolean) variable with a value of:

- 1 (true) if the SLC has acknowledged the SLC_M1_Update_Request for a file transfer from the 1394 Turbo to the SLC, and
- 0 (false) if the SLC has not acknowledged the SLC_M1_Update_Request from the 1394 Turbo controller (or if the transfer is complete).

SLC M1 Update Request

SLC_M1_Update_Request is a logical (Boolean) variable with a value of:

- 1 (true) if the 1394 Turbo controller is requesting to send M1 file data to the SLC, and
- 0 if not.

An SLC_M1_Update_Request precedes an SLC_M1_Update_Acknowledge variable.

DH-485 Status

DH485_status is an integer value representing the status of the most recent DH-485 message initiated by the motion controller.

Value	Description
2	Busy
1	Offline
0	Online

In the Tag Explorer select General System Variables, then select DH-485_status in the Tag Window.

When DH485_status = 2, the motion controller is busy doing one of the following caused by a DH-485 Value block with either Send or Read selected:

- Reading a value.
- Sending a value.
- Waiting for an acknowledgment or reply from a remote device on the DH-485 network.

When DH485_status = 1, the motion controller is not communicating on the DH-485 network. This condition is usually caused by a DH-485 fault.

When DH485_status = 0, the motion controller is communicating properly with other devices on the DH-485 network. This is the normal status condition.

Axis I Limit Status 1394

The Axis_i_limit_Status variable reports the status of the torque/current limit imposed by the controller. If the torque, being commanded by the control, is greater than the preset value of the 1394 current limit or the dynamic It limit (see below), this status variable returns the value 1.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_i_limit_status in the Tag Window.

Axis It Limit Status 1394

The Axis_it_limit_Status_1394 variable reports the status of the torque/current imposed by the it (the motor current integrated over time) limit of the controller. If more torque is being commanded by the control, exceeding the present value of the 1394 axis it limit, this status variable returns a 1. The it limit value changes dynamically as the commanded current to the axis is integrated over time. In situations demanding high duty cycle acceleration and deceleration motion profiles, the drive it limit can actually fall below the axis current limit setting in order to protect the drive from an it fault condition.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_it_limit_status in the Tag Window.

System Module Fault Status 1394

System_Module_Fault_Status_1394 is an integer value representing the highest priority fault condition currently active in the system module. The table below shows the System Module Fault 1394 Status 1394 values, and the corresponding message that appears in a System Fault 1394 field in the runtime display (if enabled).

Fault Value	Description	Runtime Display
10	System_smrt_pwr_pre-charge_ fault _ 1394	CUR FLT
9	System_smrt_pwr_i_limit_fault_1394	CUR FLT
8	System_smrt_pwr_shunt_timeout_ fault_1394	STF FLT
7	Ring Fault	RNG FLT
6	Control Power Loss	CTR FLT
5	Buss Over Voltage Fault	BOV GLT
4	Buss Loss Fault	BUS FLT
3	Phase Loss Fault	PHS FLT
2	Over Temperature Fault	TMP FLT
1	Ground Fault	GND FLT
0	No Faults	SYST OK

In the Tag Explorer select General System Variables, then select System_module_fault_status_1394 in the Tag Window.

Axis Module Fault Status 1394

Axis_module_fault_status_1394 is an integer value representing the highest priority fault condition currently active in the axis module. The table below shows the 1394 Axis Module Fault Status values, and the corresponding message that appears in an Axis Fault field in the runtime display (if enabled).

Fault Value	Description	Runtime Display
11	Ring Fault	RNG FLT
10	Not Installed	NO AXIS
9	Axis_bus_loss_fault_1394	BUS FLT
8	Axis_power_fault_1394	PWR FLT
7	Axis_it_fault_1394	It FLT
6	Axis_Motor_over_temp_fault_1394	TMP FLT
5	Motor_Thermal_1394	MTR FLT
4	Position_error_fault	ERR FLT
3	Hardware_overtravel_fault	HRD FLT
2	Software_overtravel_fault	SFT LIM
1	Encoder_loss_fault or Encoder_noise_fault	ENC FLT
0	No Faults	AXIS OK

In the Tag Explorer select both Axis System Variables and the desired physical axis, then select System_module_fault_status_1394 in the Tag Window.

System Smart Power I Limit Status 1394

System_smrt_pwr_i_limit_status_1394 is a logical (Boolean) variable, with a value of:

- 1 (True) if forward, bus charging or regenerative current reaches 80% if the Smart Power system module 34 amperes Root Mean Square (RMS) continuous limit, or
- 0 (False) if not.

System Smart Power Shunt It Disable 1394

System_smrt_pwr_shunt_it_disable_1394 is a logical (Boolean) variable, with a value of:

- 1 (True) if the Smart Power system module shunt It disable bit (which enables the System_smrt_pwr_shunt_timeout_fault_1394 fault variable) is disabled, or
- 0 (False) if this bit, and the System_smrt_pwr_shunt_timeout_fault_1394 variable, are not disabled.

Use this variable only for diagnostic purposes. Use an Equation Block to set this variable to the desired value.

Note: When power to the system is turned ON, the value of this variable is reset to 0.

System Smart Power Shunt Timeout Status 1394

System_smrt_pwr_shunt_timeout_status_1394 is a logical (Boolean) variable, with a value of:

- 1 (True) if current in the Smart Power system module shunt resistor equals or exceeds 80% of the 34 amperes Root Means Square (RMS) continuous current limit, or
- 0 (False) if not.

Diagnostic Variables

Diagnostic variables let you include axis and system conditions as elements in an expression. All diagnostic variables—except Analog_Test_Outputs_1394—are read-only (they can be used in an expression, but cannot be assigned a value). Analog_Test_Outputs_1394 can both be used in an expression and be assigned a value.

Variable	Units
Analog_test_output_0_1394	Volts
Analog_test_output_1_1394	Volts
Axis_count_1394	Number
System_kw_1394	Kwatts
System_rated_current_1394	Amps
Axis_kw_1394	Kwatts
Axis_rated_current_1394	Amps
Axis_bridge_i_limit_1394	%Irated
Axis_current_scaling_1394	0 - 15
System_smrt_pwr_bus_voltage_1394	Volts
System_smrt_pwr_mtr_pwr_%_used_1394	% of max RMS continuous current
System_smrt_pwr_pic_sftwr_ver_1394	
System_smrt_pwr_regn_pwr_%_used_1394	% of max RMS continuous current
System_smrt_pwr_shunt_pwr_%_used_1394	% of max RMS continuous current

Analog Test Output 0 1394

The Analog_test_output_0_1394 variable reports the current voltage being generated by the 1394 analog output A_TEST_0.

In the Tag Explorer select General System Variables, then select Analog_test_output_0_1394 in the Tag Window.

If you set the Analog_Test_0_Mode_1394 data parameter to 0 (using a Control Settings block with Adjust selected), you can also assign this variable a value (using an Equation block with Configured selected) that is converted to an equivalent voltage on the A_TEST_0_output_1394. The voltage range of this output is -10 to +10 Volts.

Analog Test 0 Mode 1394 (D99)	**Analog Test 0 Axis 1394 (D101)**	**Analog Test Output 1394 (System Diagnostic Variable)**
0 = Generic	N/A	User-assigned value (converted to equivalent output voltage to output A_TEST_0)
1 = Velocity	0 – 3	Voltage representation of output A_TEST_0 (1 volt = 1 KRPM)
2 = Torque	0 – 3	Voltage representation of output A_TEST_0 (1 volt = 40% Rated Torque)

The A_TEST_0 output is one of two analog outputs provided for diagnostic purposes.

Analog Test Output 1 1394

The Analog_test_output_1_1394 variable reports the current voltage being generated by the 1394 analog output A_TEST_1.

In the Tag Explorer select General System Variables, then select Analog_test_output_1_1394 in the Tag Window.

If you set the Analog_Test_1_Mode_1394 data parameter to 0 (using a Control Settings block with Adjust selected), you can also assign this variable a value (using an Equation block with Configured selected) that is converted to an equivalent voltage on the A_TEST_1_output_1394. The voltage range of this output is -10 to +10 Volts.

Analog Test 0 Mode 1394 (D100)	**Analog Test 0 Axis 1394 (D102)**	**Analog Test Output 1394 (System Diagnostic Variable)**
0 = Generic	N/A	User-assigned value (converted to equivalent output voltage to output A_TEST_1)
1 = Velocity	0 – 3	Voltage representation of output A_TEST_1 (1 volt = 1 KRPM)
2 = Torque	0 – 3	Voltage representation of output A_TEST_1 (1 volt = 40% Rated Torque)

Axis Count 1394

The Axis_Count_1394 variable reports the number of 1394 Axis Modules installed.

In the Tag Explorer select General System Variables, then select Axis_count_1394 in the Tag Window.

System kw 1394

The System_kw_1394 variable reports the kilowatt rating of the 1394 System Module.

In the Tag Explorer select General System Variables, then select System_kw_1394 in the Tag Window.

System Rated Current 1394

The System_rated_current_1394 variable reports the maximum current rating of the 1394 system module in Amps.

In the Tag Explorer select General System Variables, then select System_rated_current_1394 in the Tag Window.

Axis kw 1394

The Axis_kw_1394 variable reports the kilowatt rating of the associated 1394 Axis Module.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_kw_1394 in the Tag Window.

Axis Rated Current 1394

The Axis_rated_current_1394 variable reports the maximum current rating of the associated 1394 Axis Module in Amps.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_rated_current_1394 in the Tag Window.

Axis Bridge I Limit 1394

The Axis_bridge_i_limit_1394 variable reports the absolute current/ torque limit of the associated axis module, based on the current motor selection for that axis. If the axis module is motor-limited (smaller motor case), the bridge limit is 300%. If the axis module is drive-limited (larger motor case), the bridge limit is 200%. This 1394 controller uses this value to limit the range allowed for the 1394 current limit setup parameters.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_bridge_i_limit_1394 in the Tag Window.

Axis Current Scaling 1394

The Axis_current_scaling_1394 variable reports the configured value of the associated axis module based on the current motor selection. The control automatically sets the current scaling (16 possible settings) based on the axis current rating and motor rating selected.

In the Tag Explorer, select both Axis System Variables and the desired physical axis, then select Axis_current_scaling_1394 in the Tag Window.

System Smart Power Bus Voltage 1394

The System_smrt_pwr_bus_voltage_1394 variable reports DC bus voltage in the Smart Power system module. Voltage range is 0 to 825 Volts with accuracy to within +/- 6.44 Volts.

System Smart Power Motor Power Percent Used 1394

The System_smrt_pwr_mtr_pwr_%_used_1394 variable reports motoring (forward direction) power utilization in the Smart Power system module, as a percentage of the 34 Amperes Root Mean Square (RMS) continuous current limit.

System Smart Power PIC Software Version 1394

The System_smrt_pwr_pic_sftwr_ver_1394 defines the software version of the Programmable Interface Controller (PIC) processor in the Smart Power system module.

System Smart Power Regenerative Power Percent Used 1394

The System_smrt_pwr_regn_pwr_%_used_1394 variable reports regenerative (backward direction) power utilization in the Smart Power system module, as a percentage of the 34 Amperes Root Mean Square (RMS) continuous current limit.

System Smart Power Shunt Power Percent Used 1394

The System_smrt_pwr_shunt_pwr_%_used_1394 variable represents the sum of Smart Power system module forward (motoring) and backward (regenerative) currents, as a percentage (from 0 to 100%) of the 34 Amperes Root Mean Square (RMS) continuous current limit.

System Functions

The Expression Builder presents two types of system functions:

- Motion controller functions, which provide indirect addressing of variables, and direct access to cam profile tables and I/O
- Mathematical Functions, which provide standard algebraic and trigonometric functions for use in expressions

You include a system function in an expression by selecting **System Functions** in the Tag Explorer. In the Tag Window, select the desired function. In the Expression = window, complete the expression by inserting the argument (see below). Select OK to set the function.

Like system variables, system functions can be used as elements in expressions. Unlike variables, however, functions require an argument. The argument is the element (or expression) upon which the function is performed. For example, in the following expression the user variable Cut_Length is the argument of the sine function.:

Sine(Cut_Length)

Important: The argument for motion controller functions is enclosed in brackets, whereas the argument for mathematical functions is enclosed in parentheses.

All system functions are strings of standard ASCII characters and can be typed directly into the expression.

To aid in the use of functions in expressions, a template for proper construction of the argument appears beneath the Tag Window when you select a function. The template shows the form and units of the argument. For example, the template for the sine function indicates that the argument is an angle measured in radians:

Sine(angle in radians)

Motion Controller Functions

Motion controller functions provide indirect addressing of variables and direct access to cam profile tables and the discrete I/O of the motion controller. The available motion controller functions are listed below.

Function	Argument
Indirect_variable	Address: 0 – 999 for Integrated and Basic controllers 0 – 1999 for Compact and 1394 controllers
Indirect_DH_variable	File Type, File Number, Element, Supplement
Indirect_SLC_M0F_variable	Address 0 – 255
Indirect_SLC_M0I_variable	Address 0 – 511
Indirect_SLC_M1F_variable	Address 0 – 255
Indirect_SLC_M1F_variable	Address 0 – 511
RIO_adapter_variable	Address, Type, Scale Factor
RIO_scanner_variable	Rack, Group, Type, Scale Factor
Task_status	Task Number 0 – 9
Input	Address 0 – 11
Miscellaneous_input	Input Number 0 – 9
Axis_input	Input Number 0 – 9
RIO_input	Address 0 – 99
RIO_scanner_input	Group, Address
SLC_bit_input	Address 0 – 39
AxisLink_input	Controller Node 0 - 15, Address 0 – 15
Flex_IO_input	Module 0 - 7, Address 0 – 15
Flex_IO_analog_input	Module 0 - 7, Address 0 – 7
Group_input	Address 0 – 11, Total Signals in Group 1 - 12, Optional Mask

Function	Argument
RIO_group_input	Address 0 – 99, Total Signals in Group 1 - 16, Optional Mask
RIO_scanner_group_input	Group Address, Total Signals, Optional Mask
SLC_bit_group_input	Address 0 – 39, Total Signals in Group 1 – 16, Optional Mask
AxisLink_group_input	Controller Node, Address, Total Signals in Group, Optional Mask
Flex_IO_group_input	Module, Address, Total Signals in Group, Optional Mask
Output	Address 0 – 11
Axis_output	Address 0 – 1
RIO_output	Address 0 – 99
RIO_scanner_output	Group, Address
SLC_bit_output	Address 0 – 39
AxisLink_output	Address 0 – 15
Flex_IO_output	Module 0 - 7, Address 0 – 15
Flex_IO_analog_output	Module 0 - 7, Address 0 –3
Get_Firmware	No arguments are needed
Group_output	Address 0 – 11, Total Signals in Group 1 - 12, Optional Mask
RIO_group_output	Address 0 – 99, Total Signals in Group 1 - 16, Optional Mask
RIO_scanner_group_output	Group, Address, Total Signals in Group, Optional Mask
SLC_bit_group_output	Address 0 – 39, Total Signals in Group 1 – 16, Optional Mask
AxisLink_group_output	Controller Node, Address, Total Signals in Group, Optional mask
Flex_IO_group_output	Module, Address, Total Signals in Group, Optional Mask

Function	Argument
Flex_IO_module_type	Module Type Number
Master_CAM_position	PCAM Master Point: S Class controllers 0 – 1999 Compact and 1394 controllers 0 - 12999
Master_CAM_time	TCAM Master Point: S Class controllers 0 – 1999 Compact and 1394 controllers 0 - 12999
Slave_CAM_position	PCAM Slave Point: S Class controllers 0 – 1999 Compact and 1394 controllers 0 - 12999

Each of the motion controller functions is explained on the following pages.

Indirect Variable

Use the Indirect_variable function to let the value of one user variable or expression specify another user variable. This indirect referencing lets you use multiple user variables as arrays of values. An array is a list of values—positions—that you can use in an expression by specifying the location of the element in the list, rather than the value of the element itself.

You use the Build Table block to create arrays of user variables for use with indirect referencing.

In addition, defined user variables can be referenced indirectly by specifying the corresponding address as the argument for the Indirect_variable function.

To view the address, select User Variables in the Tag Explorer. The user address for each defined variable appears in the Tag Window's Address column. (Widen the Tag window to show the third column, if it doesn't appear at first.)

Syntax:

Indirect_variable[address]

Insert a value or expression equal to the desired user address. The argument is evaluated as an integer (floating-point values are truncated), and must have a value from 0 to 999 for Integrated and Basic controllers, or from 0 to 1999 for Compact and 1394 controllers.

For example, lets say user variables 900 through 909 store an array of axis positions, to be used depending on which product is selected by an operator. If the operator's choice is stored in the user variable, Product_Choice, as a value between 0 and 9, using the following expression for the Position value in a Move Axis block which moves the axis to position.

Indirect_variable[Product_Choice + 900]

Specifically, if the operator selects the fifth product in the list, Product_Choice = 4, and the expression uses the position stored in user variable 904 (4 + 900) for the move.

Indirect DH-485 Variable

Use the Indirect_DH_variable function to let the value of a user variable, or expression, specify the value of local DH-485 variable. This indirect referencing lets you access the elements of the motion controller's DH-485 data files, by specifying the location of the element in the file rather than the element itself.

Syntax:

Indirect_DH_variable[file type, file number, element, sub-element]

Argument	Description
File Type	The numerical value of the type of file to transfer: Binary3 Integer7 Floating8 ASCII10 BCD11 IntFloat12

Argument	Description
File Number	An integer value from 0 to 15.
Element	The selected file's specific element number. File types have the following kinds and numbers of elements: BinaryWords0 – 1023 Integer16-bit Integer values0 – 1023 FloatingFloating-Point values0 – 511 ASCIICharacters0 – 2047 BCD4-digit BCD Integers0 – 10-23 IntFloatFloating Point values0 – 511
Sub-element	If you selected Binary File Type, this is the selected element's specific bit (from 0 to 15).

Except for the type number, the argument values are identical to the corresponding values used to define the selected DH-485 local variable.

Except for binary files, the sub element argument value should always be 0.

Indirect SLC M0 Float Variable

Use the Indirect_SLC_M0_Float_variable to transfer large, non-time-critical groups of data—such as endpoints and CAM tables—from the SLC to the 1394 GMC Turbo in floating point format.

You define and select SLC M0 Floats using the Tag Explorer (under SLC M0 Float) and the Tag Window. It is not necessary to use the Indirect SLC M0 Float Variable function to specify them directly.

Syntax:

Indirect_SLC_M0_Float_variable[variable number]

Insert a value or expression equal to the desired variable number. The argument is evaluated as a floating-point value, and must have a value from 0 to 255.

Indirect SLC M0 Integer Variable

Use the Indirect_SLC_M0_Integer_variable to transfer large, non-time-critical groups of data—such as endpoints and CAM tables—from the SLC to the 1394 GMC Turbo in integer format.

You define and select SLC M0 Integers using the Tag Explorer (under SLC M0 Integer) and the Tag Window. It is not necessary to use the Indirect SLC M0 Integer Variable function to specify them directly.

Syntax:

Indirect_SLC_M0_Integer_variable[variable number]

Insert a value or expression equal to the desired variable number. The argument is evaluated as an integer, and must have a value from 0 to 511.

Indirect SLC M1 Float Variable

Use the Indirect_SLC_M1_Float_variable to transfer large, non-time-critical groups of data—such as endpoints and CAM tables—from the 1394 GMC Turbo to the SLC floating point format.

You define and select SLC M1 Floats using the Tag Explorer (under SLC M1 Float) and the Tag Window. It is not necessary to use the Indirect SLC M1 Float Variable function to specify them directly.

Syntax:

Indirect_SLC_M1_Float_variable[variable number]

Insert a value or expression equal to the desired variable number. The argument is evaluated as a floating-point value, and must have a value from 0 to 255.

Indirect SLC M1 Integer Variable

Use the Indirect_SLC_M1_Integer_variable to transfer large, non-time-critical groups of data—such as endpoints and CAM tables—from the 1394 GMC Turbo to the SLC in integer format.

You define and select SLC M1 Integers using the Tag Explorer (under SLC M1 Integer) and the Tag Window. It is not necessary to use the Indirect SLC M1 Integer Variable function to specify them directly.

Syntax:

Indirect_SLC_M1_Integer_variable[variable number]

Insert a value or expression equal to the desired variable number. The argument is evaluated as an integer, and must have a value from 0 to 511.

RIO Adapter Variable

Use the RIO_adapter_variable function to specify RIO adapter formatted data input groups directly in an expression.

You define and select RIO adapter inputs using the Tag Explorer (under RIO Formatted Input) and the Tag Window. It is not necessary to use the RIO Adapter Variable function to specify them directly.

Syntax:

RIO_adapter_variable[address, type, scale factor]

Argument	Description
Address	The beginning address: 4, 20, 36 or 68.
Type	One of the following numerical values: 0:Formatted Binary 1:Formatted BCD
Scale factor	If you use the scale factor, the controller will multiply this number against the value read from the PLC output image table. If you do not use scale formatted data when defining this variable, the value of the formatted data input group equals the value read from the PLC output image table (effectively using a scale factor of 1).

See *Using the RIO Adapter Option* chapter for more details.

RIO Scanner Variable

Use the RIO_scanner_variable function to directly specify RIO scanner formatted data input groups in an expression.

You define and select RIO scanner inputs using the Tag Explorer (under RIO Formatted Input) and the Tag Window. It is not necessary to use the RIO Scanner Variable function to specify them directly.

Syntax:

RIO_scanner_variable[rack, group, type, scale factor]

Argument	Description
Rack	The rack address of the controller (from 0 to 7) from which I/O is read.
Group	The starting group of the selected rack's discrete I/O to be read.
Type	One of the following numerical values: 0:Formatted Binary 1:Formatted BCD
Scale factor	If you use the scale factor, the controller will multiply this number against the value read from the PLC output image table. If you do not use scale formatted data when defining this variable, the value of the formatted data input group equals the value read from the PLC output image table (effectively using a scale factor of 1).

See your motion controller's *Installation and Setup* manual for further information.

Task Status

Task_status is an integer value representing the current state of the specified task in the motion controller. It allows mathematical or logical calculations to be made based on the status of a task.

You use an On Task block (with either If Task or Wait for Task selected) in the GML Commander diagram to check the Task_status function, rather than use the Task Status function.

Syntax:

Task_status[task number]

Type an integer value or expression, from 0 to 9 inclusive, representing the task number you want to check.

The table below shows the Task_status function's returnable values, and the status indicated by each value.

Value	Status
2	Task suspended
1	Task running
0	Task stopped or never started

A task is:

- suspended, if it has been paused by a Stop Dispatcher command in the Task Control block.
- running, after it has been started or resumed with a Task Control block
- not running, if it has not been started or has been stopped by a Stop Current Task or Stop Other Task block.

Input

The Input function lets you directly specify the general purpose discrete inputs on the motion controller in an expression, instead of using their defined names. This function is available only to Integrated and Basic controllers. (It is not available to 1394 or Compact controllers.)

Syntax:

Input[address]

Insert a value or expression equal to the desired address. The argument is evaluated as an integer (floating point values are truncated) and must have a value from 0 to 11 inclusive. The Input function evaluates as 1 (true) if the input is ON, and 0 (false) if it is OFF.

Miscellaneous Inputs

Use the Miscellaneous Input function to include the state of the non-axis-specific, dedicated input on the motion controller in an expression.

Syntax:

Miscellaneous_input[input number]

Insert a value or expression equal to the desired input number from the table below.

Input Number	Description
0	Initialization Input
1	Application Lock
2	Data Lock (Compact, Integrated, and Basic only)
3	Plug Lock
4	External Enable 1394 (1394 only)
5	Hard Fault 1394 (1394 only)
6	Watchdog OK 1394 (1394 only)
7	Dualport Initialization 1394 (1394 only)
8	DSP Bus Grant
9	DH485 Switch

The argument is evaluated as an integer (floating point values are truncated) and must have a value from 0 to 9 inclusive.

The Miscellaneous Input function is evaluated as 1 (true) if the input is ON and 0 (false) if it is OFF. The Miscellaneous Input function reads the current physical state (ON/OFF) of the input.

Axis Input

Use the Axis_input function to include the state of the motion controller's axis specific, dedicated, discrete inputs in an expression. After selecting Axis_input in the Expression Builder's Tag Window, be sure to select the desired Axis in the pop-up menu.

Syntax:

Axis_input[input number]

Insert a value or expression equal to the desired input number from the table below for the argument.

Input Number	Description
0	Home Input
1	Positive Overtravel Input
2	Negative Overtravel Input
3	Drive Fault Input
4	Registration Input
5	Encoder Loss State (1394 and Compact only)
6	Marker State
7	CXIC Interrupt State
8	Resolver Loss Event (1394 only)
9	Resolver Loss State (1394 only)

Important: Do not attempt to use an argument your motion controller does not support. A runtime fault results.

The argument is evaluated as an integer (floating point values are truncated) and must have a value between 0 and 9 inclusive. The Axis_input function is evaluated as 1 (true) if the input is ON and 0 (false) if it is OFF. The Axis_input function reads the current physical state (ON or OFF) of the input. The Contacts? setting (NORMAL OPEN or NORMAL CLOSED) in the motion controller's machine setup menu is ignored for the overtravel and drive fault inputs.

RIO Input

Use the RIO_input function to directly specify the RIO adapter inputs on IMC-S/2xx-R model motion controllers in an expression.

You define and select RIO adapter inputs using the Tag Explorer (under RIO Adapter Input) and the Tag Window. It is not necessary to use the RIO Adapter Input function to specify them directly.

Syntax:

RIO_input[address]

Insert a value or expression equal to the desired input address for the argument. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from 0 to 99 inclusive. The user-defined RIO adapter inputs are numbered, and correspond to PLC output bits as shown in the following table:

User-Defined Discrete Inputs	**Correspond to PLC Outputs in I/O Group…**			
0 – 3 *	1 3 5 7	1 3 5	1 3	1
4 – 19	Not Available	2 4 6	2 4	2
20 – 35	Not Available	3 5 7	3 5	3
36 – 51	Not Available	N/A	4 6	4
52 – 67	Not Available	N/A	5 7	5
68 – 83	Not Available	N/A	N/A	6
84 - 99	Not Available	N/A	N/A	7

User-Defined Discrete Inputs	Correspond to PLC Outputs in I/O Group…			
Starting Group	0 2 4 6	0 2 4	0 2	0
Rack Size	1/4	1/2	3/4	Full

* User-defined discrete inputs 0 – 3 correspond to PLC output bits 14 – 17 (octal) in the appropriate I/O group.

The RIO_input function is evaluated as 1 (true) if the input is ON and 0 (false) if it is OFF.

RIO Scanner Input

Use the RIO_scanner_input function to directly specify the RIO scanner inputs on IMC-S/2xx-R model motion controllers in an expression. The RIO_scanner_input function is not available for 1394 or Compact motion controllers.

You define and select RIO scanner inputs in the Tag Explorer (under RIO Scanner Input) and Tag Window. It is not necessary to use the RIO scanner input function to specify them directly.

Syntax:

RIO_scanner_input[group, address]

Argument	Description
Group	The starting group of the discrete I/O to be read. (Take this from the RIO Scanner Input dialog box, shown below.)
Address	The address, or bit number, of the selected starting group to be read. (Take this from the RIO Scanner Input dialog box, shown below.)

The RIO_scanner_input function is evaluated as 1 (true) if the input is ON and 0 (false) if it is OFF.

SLC Bit Input

Use the SLC_bit_input function to directly specify the SLC input bits on 1394 motion controllers in an expression. The SLC_bit_input function is not available for Compact, Integrated, or Basic motion controllers.

You define and select SLC input bits in the Tag Explorer (under SLC Input Bit) and Tag Window. It is not necessary to use the SLC_bit_input function to specify them directly.

Syntax:

SLC_bit_input[address]

Insert a value or expression equal to the desired input address for the argument. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from 0 to 39 inclusive.

AxisLink Input

Use the AxisLink_input function to directly specify AxisLink inputs to IMC-S/2xx-L model motion controllers in an expression

You define and select AxisLink inputs using the Tag Explorer (under AxisLink I/O Input) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

AxisLink_input[controller node, address]

Argument	Description
Controller Node	The controller node on the link (from 0 to 15) from which output is read.
Address	The address, or signal number, (from 0 to 15) of the selected controller's specific output to be read.

The AxisLink_input function is evaluated as 1 (true) if the input is ON and 0 (false) if it is OFF.

Flex I/O Input

Use the Flex_IO_input function to directly specify Flex I/O discrete inputs in an expression. This function is available only if your motion controller is running iCODE version 3.0 or later (as selected in the General page of the Configure Control Options dialog box).

You define and select Flex I/O discrete inputs in the Tag Explorer (under Flex I/O Input) and the Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Flex_IO_input[module, address]

Argument	Description
Module	The Flex I/O module (from 0 to 7) from which output is read.
Address	The address of the selected module's specific output (from 0 to 15) to be read.

The Flex_IO_input function is evaluated as 1 (true) if the input is ON and 0 (false) if it is OFF.

Flex I/O Analog Input

Use the Flex_IO_analog_input function to directly specify Flex I/O analog inputs in an expression. This function is available only if your motion controller is running iCODE version 3.0 or later (as selected in the General page of the Configure Control Options dialog box).

You define and select Flex I/O analog inputs in the Tag Explorer (under Flex I/O Analog Input) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Flex_I/O_analog_input[module, address]

Argument	Description
Module	The Flex I/O module number (from 0 to 7) whose output is read.
Address	The address of the selected module's specific input (from 0 to 15) to be read.

Group Input

Use the Group_input function to directly specify discrete input groups (Integrated and Basic only) in an expression.

You define and select input groups in the Tag Explorer (under General Purpose I/O Input Group) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Group_input[address, total signals in group, optional mask]

Argument	Description
Address	The address or signal number (from 0 to 11) of the first signal from which output is read.
Total Signals in Group	The total number of consecutive signals comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks*.)

Optional Masks

Use a bit mask to exclude or mask unwanted inputs (bits) from the input group. Using a bit mask lets you create an input group consisting of non-contiguous inputs. For example, you can use a bit mask to create a group consisting of four inputs, starting at input #2 and running through input #5, but excluding input #3.

The bit mask is the sum of the binary value of each group input bit whose signal is read. The binary value of each group input bit whose signals are not read are excluded from this sum. The following table lists the binary values for up to 16 bits:

For this bit in an input group	Use this Binary Value
1st	1
2nd	2
3rd	4
4th	8
5th	16
6th	32
7th	64
8th	128
9th	256
10th	512
11th	1,024
12th	2,048
13th	4,096
14th	8,192
15th	16,384

For this bit in an input group	Use this Binary Value
16th	32,768

In our example, to create a bit mask for the group beginning at bit #2 and ending at bit #5, but excluding bit #3, add together these binary values:

Bit Number / Status	Binary Value
2 / ON	1 (the first bit in the group)
3 / OFF	0 (masked)
4 / ON	4 (the third bit in the group)
5 / ON	8 (the fourth bit in the group)

Bit Mask = 13

The first input in your group (unless it is masked) has the binary value of 1, no matter what its address number may be.

RIO Group Input

Use the RIO_group_input function to directly specify RIO adapter input groups in an expression.

You define and select input groups using the Tag Explorer (under RIO Adapter Input Group) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

RIO_group_input[address, total signals in group, optional mask]

Argument	Description
Address	The address or signal number (from 0 to 11) of the first signal whose output is read.

Argument	Description
Total Signals in Group	The total number of consecutive signals (from 0 to 16) comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks* in the *Group Input* section.)

RIO Scanner Group Input

Use the RIO_scanner_group_input function to directly specify RIO scanner input groups in an expression.

You define and select input groups using the Tag Explorer (under RIO Scanner Input Group) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

RIO_scanner_group_input[group, address, total signals, optional mask]

Argument	Description
Group	The starting group of the input (from 0 to 15) to be read.
Address	The address or signal number (from 0 to 11), of the first signal from which input is read.
Total Signals	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks* in the *Group Input* section.)

SLC Bit Group Input

Use the SLC_bit_group_input function to directly specify SLC bit input groups in an expression.

You define and select input groups using the Tag Explorer (under SLC Input Group Bit) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

SLC_bit_group_input[address, total signals in group, optional mask]

Argument	Description
Address	The address or signal number (from 0 to 39) of the first signal whose output is read.
Total Signals in Group	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks* in the *Group Input* section.)

AxisLink Group Input

Use the AxisLink_group_input function to directly specify AxisLink input groups in an expression.

You define and select input groups in the Tag Explorer (under AxisLink I/O Input Group) and the Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

AxisLink_group_input[controller node, address, total signals in group, optional mask]

Argument	Description
Controller Node	The controller node on the link (from 0 to 15), from which output is read.
Address	The address, or signal number (from 0 to 15), of the first signal from which output is read.

Argument	Description
Total Signals in Group	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks* in the *Group Input* section.)

Flex I/O Group Input

Use the Flex_IO_group_input function to directly specify Flex I/O discrete input groups in an expression. The Flex_I/O_group_input function is available only if your motion controller is running iCODE version 3.0 or later (as selected in the Configure Control Options dialog box).

You define and select input groups in the Tag Explorer (under Flex I/O Input Group) and Tag Window. It is not necessary to use this function to specify them directly

Syntax:

Flex_IO_group_input[module, address, total signals in group, optional mask]

Argument	Description
Module	Select the Flex I/O module (from 0 to 8), from which output is read.
Address	Select the address of the selected module's specific output (from 0 to 15) to be read.
Total # of Signals in the Group	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Mask* in the *Group Input* section, above.)

Output

Use the Output function to directly specify the motion controller's general purpose discrete outputs (Integrated and Basic only) in an expression.

You define and select outputs in the Tag Explorer (under General Purpose I/O Output) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Output[address]

Insert a value or expression equal to the desired output address. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from 0 to 11.

The Output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

Axis Output

Use the Axis_output function to directly specify the state of the motion controller's dedicated, axis-specific discrete outputs in an expression.

Syntax:

Axis_output[address]

Insert a value or expression equal to the desired address (or output number) from the table below for the argument:

Address	Description
0	Drive Enable Output
1	Absolute Position Strobe Output

The argument is evaluated as an integer (floating-point values are truncated) and must have a value of 0 or 1. The Axis_output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

RIO Output

Use the RIO_output function to directly specify RIO adapter outputs in an expression. This function is available only for Integrated and Basic motion controllers.

You define and select RIO adapter outputs in the Tag Explorer (under RIO Adapter Output) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

RIO_output[address]

Insert a value or expression equal to the desired output address. The argument is evaluated as an integer (floating-point values are truncated) and must have a value between 0 and 99 inclusive. The available user-defined RIO adapter outputs are numbered, and correspond to PLC output bits as shown in the following table:

User-Defined Discrete Outputs	**Correspond to PLC Inputs in I/O Group…**			
0 – 3 *	1 3 5 7	1 3 5	1 3	1
4 – 19	Not Available	2 4 6	2 4	2
20 – 35	Not Available	3 5 7	3 5	3
36 – 51	Not Available	N/A	4 6	4
52 – 67	Not Available	N/A	5 7	5
68 – 83	Not Available	N/A	N/A	6
84 - 99	Not Available	N/A	N/A	7
Starting Group	0 2 4 6	0 2 4	0 2	0
Rack Size	1/4	1/2	3/4	Full

The RIO_output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

RIO Scanner Output

Use the RIO_scanner_output function to directly specify the RIO scanner outputs, on Integrated and Basic motion controllers, in an expression.

You define and select RIO scanner outputs using the Tag Explorer (under RIO Scanner Output) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

RIO_scanner_output[group, address]

Argument	Description
Group	The starting group of the output (from 0 to 7) to be sent.
Address	The address or signal number (from 0 to 15) of the first signal from which output is sent.

The RIO_scanner_output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

SLC Bit Output

Use the SLC_bit_output function to directly specify the SLC bit outputs in an expression. This function is available only on 1394T motion controllers.

You define and select SLC bit outputs using the Tag Explorer (under SLC Output Bit) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

SLC_bit_output[address]

Insert a value or expression equal to the desired address for the argument. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from 0 to 39.

AxisLink Output

Use the AxisLink_output function to directly specify the AxisLink outputs, on IMC-S/2xx-L model motion controllers, in an expression.

You define and select AxisLink outputs using the Tag Explorer (under AxisLink I/O Output) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

AxisLink_output[address]

Insert a value or expression equal to the desired address for the argument. The argument is evaluated as an integer (floating point values are truncated) and must have a value from 0 to 15.

The AxisLink_output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

Flex I/O Output

Use the Flex_IO_output function to directly specify Flex I/O discrete outputs in an expression. This function is available only if you are running iCODE version 3.0 or later (as selected in the Configure Control Options dialog box).

You define and select Flex I/O discrete outputs using the Tag Explorer (under Flex I/O Output) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Flex_IO_output[module, address]

Argument	Description
Module	The Flex I/O module number (from 0 to 7) from which output is sent.
Address	The address of the selected module's specific output (from 0 to 15) to be sent.

The Flex_IO_output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

Flex I/O Analog Output

Use the Flex_IO_analog_output function to directly specify Flex I/O analog outputs in an expression. This function is available only if you are running iCODE version 3.0 or later (as selected in the Configure Control Options dialog box).

You define and select Flex I/O analog outputs using the Tag Explorer (under Flex I/O Analog Output) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Flex_IO_analog_output[module, address]

Argument	Description
Module	The Flex I/O module number (from 0 to 7) from which output is sent.
Address #	The address of the selected module's specific output (from 0 to 4) to be sent.

The Flex_IO_output function is evaluated as 1 (true) if the output is ON and 0 (false) if it is OFF.

Get Firmware

Use the Get_Firmware function to return the current version of firmware used in the motion controller. The returned value will appear as a string of three numerical characters, as follows:

If the firmware version is:	Get_Firmware returns:
3.9	390
3.9A	391
3.9B	392

Syntax:

Get_Firmware[ ]

Note: Unlike other system functions, no argument is required for the Get_Firmware function

Get_Firmware is available only for firmware versions 3.9 and higher.

Group Output

Use the Group_output function to directly specify general purpose (Integrated and Basic only) discrete output groups in an expression.

You define and select output groups using the Tag Explorer (under General Purpose I/O Output Group) and Tag Window.

Syntax:

Group_output[address, total signals in group, optional mask]

Argument	Description
Address	The address, or signal number (from 0 to 11), of the first signal from which output is sent.
Total Signals in Group	The total number of consecutive signals comprising the group, including any masked (or omitted) signals.

Argument	Description
Optional Mask	An integer representing the sum of the binary value of each group output bit whose signal is sent, excluding—or masking—the binary value of every group output bit whose signal is not sent. (See *Optional Masks* in the *Group Input* section.)

RIO Group Output

Use the RIO_group_output function to directly specify RIO adapter output groups in an expression.

You define and select output groups using Tag Explorer (under RIO Adapter Output Group) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

RIO_group_output[address, total signals in group, optional mask]

Argument	Description
Address	The address, or signal number (from 0 to 15), of the first signal from which input is sent.
Total Signals in Group	The total number of consecutive signals comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group output bit whose signal is sent, excluding—or masking—the binary value of every group output bit whose signal is not sent. (See *Optional Masks* in the *Group Input* section.)

RIO Scanner Group Output

Use the RIO_scanner_group_output function to directly specify RIO scanner output groups in an expression.

You define and select output groups using Tag Explorer (under RIO Scanner Output Group) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

RIO_scanner_group_output[group, address, total signals in group, optional mask]

Argument	Description
Group	The starting group of the output (from 0 to 7) to be sent.
Address	The address, or signal number (from 0 to 15), of the first signal from which output is sent.
Total Signals in Group	The total number of consecutive signals comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group output bit whose signal is sent, excluding—or masking—the binary value of every group output bit whose signal is not sent. (See *Optional Masks* in the *Group Input* section.)

SLC Bit Group Output

Use the SLC_bit_group_output function to directly specify SLC bit output groups in an expression. This function is available only with 1394 motion controllers.

You define and select output groups using Tag Explorer (under SLC Output Group Bit) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

SLC_bit_group_output[address, total signals in group, optional mask]

Argument	Description
Address	The address, or signal number (from 0 to 39), of the first signal from which output is read.
Total Signals in Group	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.

Argument	Description
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks* in the *Group Input* section.)

AxisLink Group Output

Use the AxisLink_group_output function to directly specify AxisLink output groups in an expression.

You define and select output groups using the Tag Explorer (under AxisLink I/O Output Group) and the Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

AxisLink_group_output[address, total signals in group, optional mask]

Argument	Description
Address	The address, or signal number (from 0 to 15), of the first signal from which output is read.
Total Signals in Group	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.
Optional Mask	An integer representing the sum of the binary value of each group input bit whose signal is read, excluding—or masking—the binary value of every group input bit whose signal is not read. (See *Optional Masks* in the *Group Input* section.)

Flex I/O Group Output

Use the Flex_IO_group_output function to directly specify Flex I/O discrete output groups in an expression directly. This function is available only if your motion controller is running iCODE version 3.0 or later (as selected in the Configure Control Options dialog box).

You define and select Flex I/O output groups using the Tag Explorer (under Flex I/O Output Group) and Tag Window. It is not necessary to use this function to specify them directly.

Syntax:

Flex_IO_group_output[module, address, total signals in group, optional mask]

Argument	Description
Module	Select the Flex I/O module number (from 0 to 7) from which output is sent.
Address	Select the address of the selected module's specific output (from 0 to 15) to be sent.
Total # of Signals in the Group	The total number of consecutive signals (from 1 to 16) comprising the group, including any masked (or omitted) signals.
Optional Bit Mask	An integer representing the sum of the binary value of each group output bit whose signal is sent, excluding—or masking—the binary value of every group output bit whose signal is not sent. (See *Optional Mask* in the *Group Input* section.)

Flex I/O Module Type

Use the Flex_IO_module_type function to directly determine the module type of a particular Flex IO module. This function is available only if your motion controller is running iCODE version 3.0 or later (as selected in the Control Options definition). This function provides a read-only return value. You cannot use this value to configure or assign Flex IO modules.

You configure Flex I/O modules in the Configure Flex I/O Module Use dialog box (after first enabling a module in the Flex I/O page of the Configure Control Options dialog box). It is not necessary to use this function to use Flex IO modules.

Syntax:

Flex_IO_module_type[module number]

This system function returns a numeric value based upon the configured Flex IO module installed at the specified module location (0 to 7) as shown in following table:

Flex I/O Module Type	Module Type Number
1794-IB16	10
1794-IE8	11
1794-IAS	12
1794-IF41	13
1794-OB16	64
1794-OE4	65
1794-OA8	66
1794-OW8	67
1794-0B16P	68
1794-IE 4X0E2	128
1794-IB10X0B6	129

Master Cam Position

Use the Master_CAM_position function to store a master axis position in the cam table, or use it as an element in an expression.

You use the Build Table or Equation block to build position cam master tables. It is not necessary to use this function for this purpose.

Syntax:

Master_Cam_position[PCAM master point]

After you select an axis from the pop-up menu, insert a value or expression equal to the desired position cam table master point for the argument. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from:

- 0 to 1999 for firmware versions under 3.0 (Integrated and Basic motion controllers)
- 0 to 12999 for firmware versions 3.0 and higher (1394 and Compact controllers).

Master Cam Time

Use the Master_CAM_Time function to store a master axis time point in the cam table, or use it as an element in an expression.

You use the Build Table or Equation block to build time cam master tables. It is not necessary to use this function for this purpose.

Syntax:

Master_CAM_Time[TCAM master point]

After you select an axis from the pop-up menu, insert a value or expression equal to the desired time cam table master point for the argument. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from:

- 0 to 1999 for firmware versions under 3.0 (Integrated and Basic motion controllers)
- 0 to 12999 for firmware versions 3.0 and higher (1394 and Compact controllers).

Slave Cam Position

Use the Slave_CAM_position function to store a slave axis position in the cam table, or use it as an element in an expression.

You use the Build Table or Equation block to build cam slave position tables. It is not necessary to use this function for this purpose.

Syntax:

Slave_CAM_position[PCAM slave point]

After you select an axis from the pop-up menu, insert a value or expression equal to the desired position cam slave table point for the argument. The argument is evaluated as an integer (floating-point values are truncated) and must have a value from:

- 0 to 1999 for firmware versions under 3.0 (Integrated and Basic motion controllers)
- 0 to 12999 for firmware versions 3.0 and higher (1394 and Compact controllers).

Mathematical Functions

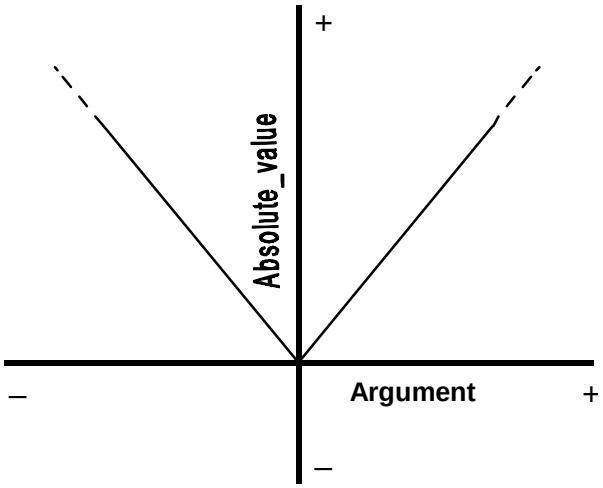
Mathematical functions provide standard algebraic and trigonometric functions for use in expressions. The available mathematical functions are shown in the table below.

Function	Argument
Absolute_value	Floating Point Value
Arc_cosine	-1 ≤ Floating Point Value ≤ 1
Arc_sine	-1 ≤ Floating Point Value ≤ 1
Arc_tangent	Floating Point Value
Cosine	Floating Point Value in Radians
Sine	Floating Point Value in Radians
Tangent	Floating Point Value in Radians
Round	Floating Point Value
Round-up	Floating Point Value
Round_down	Floating Point Value
Integer	Floating Point Value
Exponent	Floating Point Value
Log	Floating Point Value > 0
Square_root	Floating Point Value ≥ 0

All mathematical functions are read-only—they can be used within an expression to perform the desired function, but they cannot be assigned a value. Each mathematical function is explained below.

Absolute Value

The Absolute_value function is a positive value equal to the magnitude of the argument, as shown below.



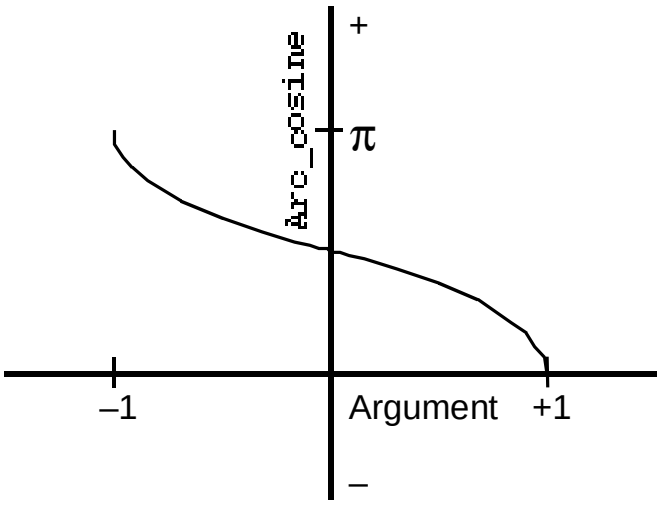
The argument can be any floating point value or expression. If the argument is positive, it is not affected by the Absolute_value function.

For example, the following expression is evaluated as 10.257 if the user variable Cut_Length has a value of 10.257 or -10.257.

Absolute_value(Cut_Length)

Arc Cosine

The value of the Arc_cosine function is the principal angle (in radians) whose cosine is equal to the argument, as shown below.



The argument can be any floating point value or expression, but it must have a value from -1 to 1, inclusive, because the cosine of an angle is always within this range. The value of the Arc_cosine function is always an angle from 0 to π radians (180°), inclusive. If the argument is not between ±1, the value of the Arc_cosine function is zero.

For example, the following expression is evaluated as 1.047*xx* since the principal angle whose cosine is 0.5 is 1.047 radians (60°).

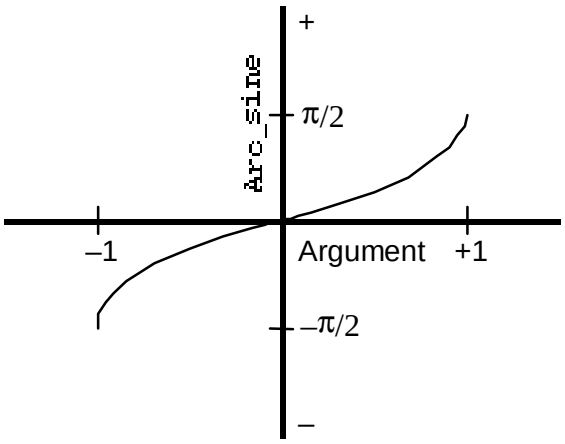
```
Arc_cosine(0.5)
```

To convert the value of the Arc_cosine function to degrees, multiply by $^{180}/\pi$. For example, the following expression is evaluated as 60 (assuming the user constant `Pi` has been defined as the value of π) since the principal angle whose cosine is 0.5 is 60°.

```
(Arc_cosine(0.5) * 180 / Pi)
```

Arc Sine

The value of the Arc_sine function is the principal angle (in radians) whose sine is equal to the argument, as shown below.



The argument can be any floating point value or expression, but it must have a value from -1 to 1, inclusive, because the sine of an angle is always within this range. The value of the Arc_sine function is always an angle from $-\pi/2$ to $\pi/2$ radians (±90°), inclusive. If the argument is not between ±1, the value of the Arc_sine function is zero.

For example, the following expression is evaluated as 0.5235*xx* since the principal angle whose sine is 0.5 is 0.5235 radians (30°).

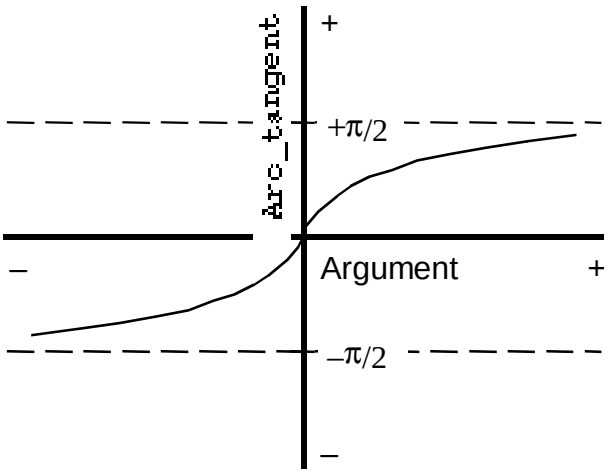
```
Arc_sine (0.5)
```

To convert the value of the Arc_sine function to degrees, multiply by $180/\pi$. For example, the following expression is evaluated as 30 (assuming the user constant `Pi` has been defined as the value of π) since the principal angle whose sine is 0.5 is 30°.

```
(Arc_sine(0.5) * 180 / Pi)
```

Arc Tangent

The value of the Arc_tangent function is the principal angle (in radians) whose tangent is equal to the argument, as shown below.



The argument may be any floating point value or expression. The value of the Arc_tangent function is always an angle from $-\pi/2$ to $\pi/2$ radians (±90°), inclusive.

For example, the following expression is evaluated as 0.7853*xx* since the principal angle whose tangent is 1 is 0.7853 radians (45°).

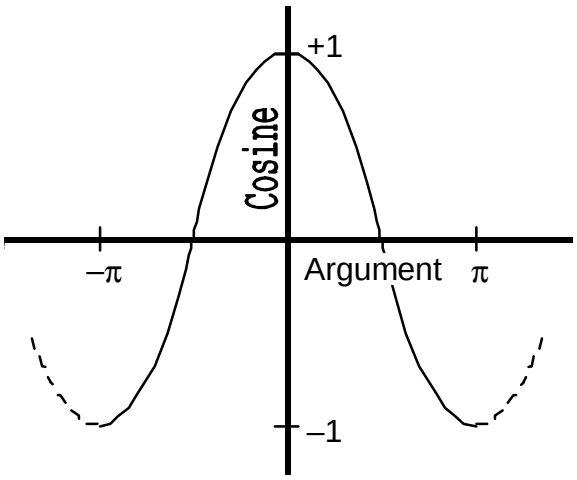
```
Arc_tangent(1)
```

To convert the value of the Arc_tangent function to degrees, multiply by $180/\pi$. For example, the following expression is evaluated as 45 (assuming the user constant `Pi` has been defined as the value of π) since the principal angle whose tangent is 1 is 45°.

```
Arc_tangent(1) * 180 / Pi)
```

Cosine

The value of the Cosine function is the cosine of the argument angle (in radians), as shown below.



The argument can be any floating point value or expression in radians. The value of the Cosine function is always from –1 to +1, inclusive.

For example, the following expression is evaluated as 0.5 since the cosine of 1.047 radians (60°) is 0.5.

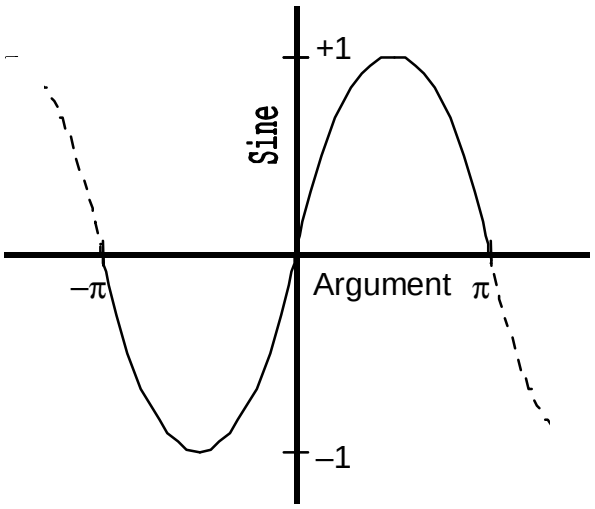
```
Cosine(1.047)
```

To convert the argument of the `Cosine` function from degrees to radians, multiply by $\pi/180$. For example, the following expression is evaluated as 0.5 (assuming the user constant `Pi` has been defined as the value of π) since the cosine of 60° is 0.5.

```
Cosine(60 * Pi / 180)
```

Sine

The value of the Sine function is the sine of the argument angle (in radians), as shown below.



The argument may be any floating point value or expression in radians. The value of the `Sine` function is always from –1 to +1, inclusive.

For example, the following expression is evaluated as 0.5 since the sine of 0.5235 radians (30°) is 0.5.

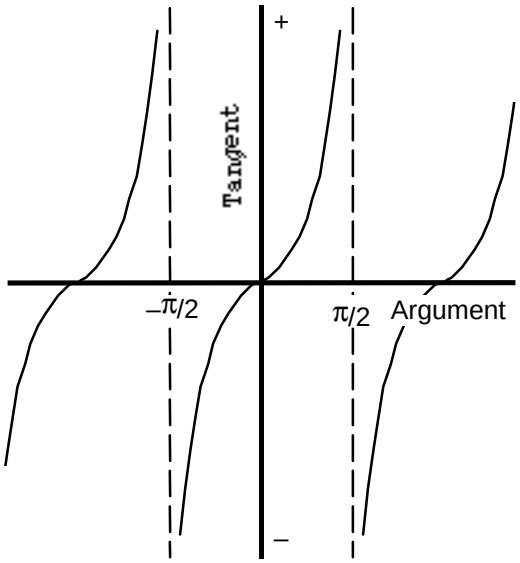
```
Sine(0.5235)
```

To convert the argument of the `Sine` function from degrees to radians, multiply by $\pi/180$. For example, the following expression is evaluated as 0.5 (assuming the user constant `Pi` has been defined as the value of π) since the sine of 30° is 0.5.

```
Sine(30 * Pi / 180)
```

Tangent

The value of the Tangent function is the tangent of the argument angle (in radians), as shown below.



The argument can be any floating point value or expression, but it must not be a multiple of $\pm\pi/2$ radians (±90°), since the tangent of $\pm\pi/2$ radians is ±∞, and thus undefined.

For example, the following expression is evaluated as 1 since the tangent of 0.7853 radians (45°) is 1.

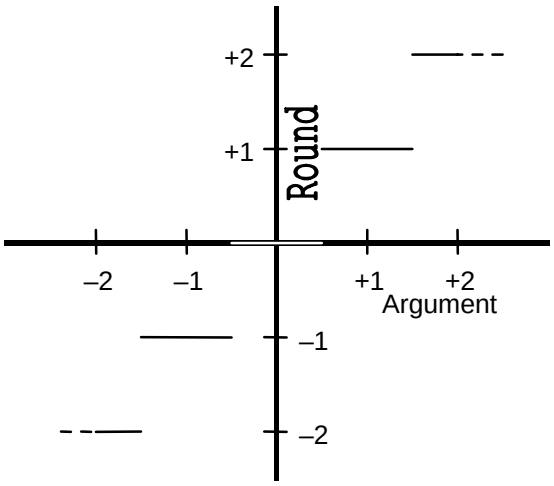
```
Tangent(0.7853)
```

To convert the argument of the Tangent function from degrees to radians, multiply by $\pi/180$. For example, the following expression is evaluated as 1 (assuming the user constant `Pi` has been defined as the value of π) since the tangent of 45° is 1.

```
Tangent(45 * Pi / 180)
```

Round

The Round function rounds the argument up or down to the nearest integer value, as shown below.



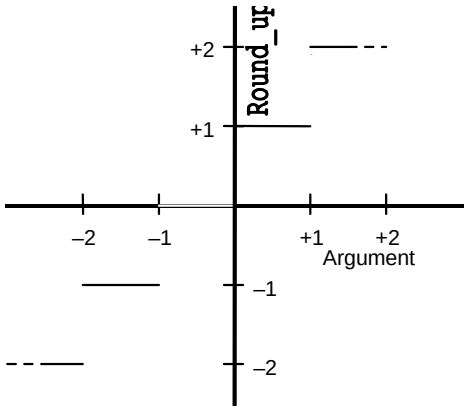
The argument may be any floating-point value or expression.

For example, the following expression is evaluated as 10 if the user variable Cut_Length has a value between 9.5 and 10.49 (9.5 ≤ Cut_Length < 10.49), 11 if Cut_Length has a value between 10.5 and 11.49 (10.5 ≤ Cut_Length < 11.49), etc.

Round(Cut_Length)

Round Up

The Round_up function rounds the argument up to the next higher integer value, as shown below.



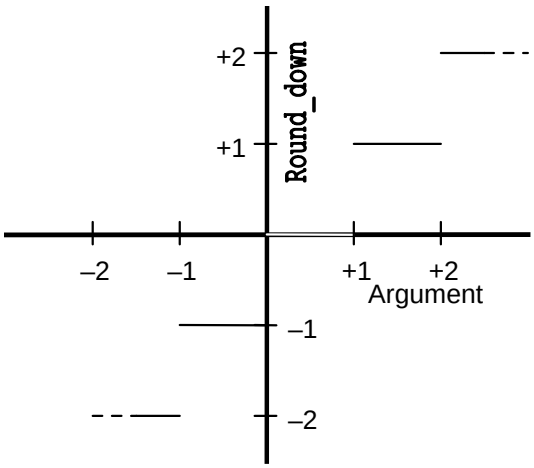
The argument can be any floating point value or expression.

For example, the following expression is evaluated as 10 if the user variable `Cut_Length` has a value between 9 and 10 (9 < Cut_Length ≤ 10), 11 if Cut_Length has a value between 10 and 11 (10 < Cut_Length ≤ 11), etc.

Round_up(Cut_Length)

Round Down

The Round_down function rounds the argument down to the next lower integer value, as shown below.



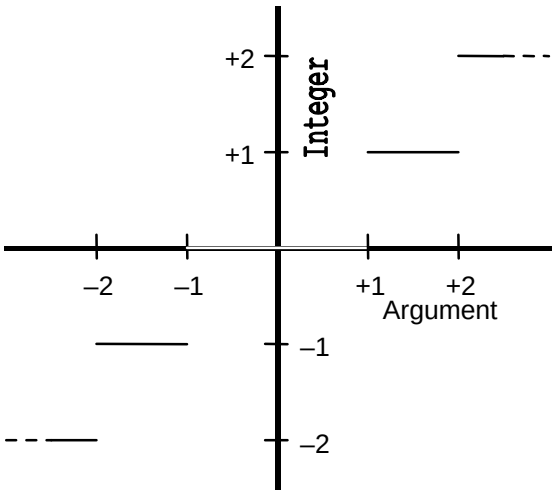
The argument can be any floating point value or expression.

For example, the following expression is evaluated as 9 if the user variable Cut_Length has a value between 9 and 10 ($9 \leq$ Cut_Length < 10), 10 if Cut_Length has a value between 10 and 11 ($10 \leq$ Cut_Length < 11), etc.

Round_down(Cut_Length)

Integer Value

The value of the Integer function is the integer part of the argument, as shown below.



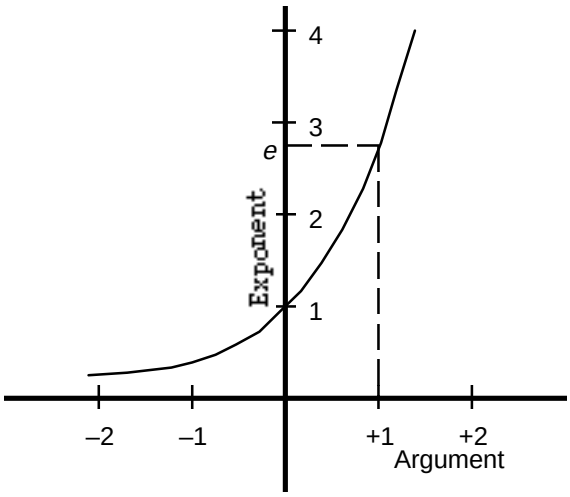
The argument can be any floating point value or expression. Note that for positive arguments, the action of the Integer function is identical to that of the Round_down function, while for negative arguments, it is identical to the Round_up function.

For example, the following expression is evaluated as 9 if the user variable Cut_Length has a value between 9 and 10 ($9 \leq$ Cut_Length < 10), -10 if Cut_Length has a value between -10 and -11 ($-11 <$ Cut_Length ≤ -10), etc.

Integer(Cut_Length)

Exponent

The value of the Exponent function is e raised to the power of the argument (e^x), as shown below.



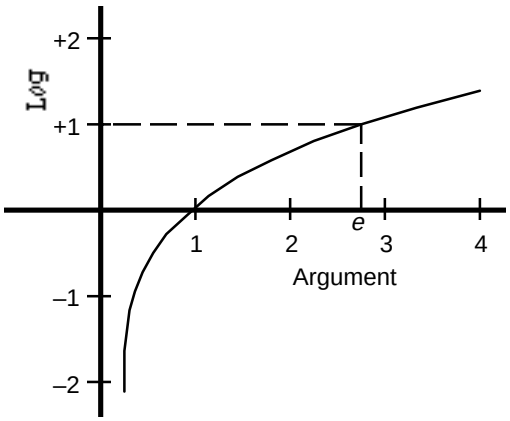
The argument may be any floating-point value or expression. The constant e is the base of the natural system of logarithms. It is an irrational number, which to 15 decimal places, is equal to 2.71828 18284 59045.

For example, the following expression is evaluated as 2.7182*xx* (e) since anything raised to the power of 1 is itself.

Exponent(1)

Natural Logarithm

The value of the Log function is the natural logarithm ($\log_e$ or ln) of the argument, as shown below.



The base of the Log function is e, is an irrational number, which to 15 decimal places, is equal to 2.71828 18284 59045.

The argument can be any floating point value or expression, but must be greater than zero. A negative or zero argument gives a value of 0 for the Log function.

For example, the following expression is evaluated as 0.693*xx* since ln 2 = 0.6931.

```
Log(2)
```

Note that the Log function is the inverse of the Exponent function. In other words, for all values of x:

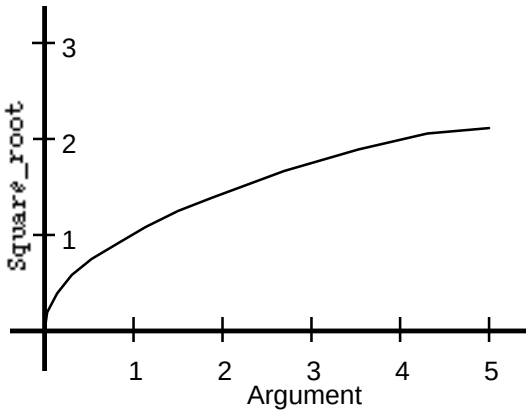
```
Log(Exponent(x)) = x
```

To find the common logarithm (base 10 logarithm or $\log_{10}$) of an argument, divide the Log function by ln 10. For example, the following expression is evaluated as 0.3010*xx* since $\log_{10}$ 2 = 0.3010.

```
Log(2) / Log(10)
```

Square Root

The value of the Square_root function is the square root ($\sqrt{}$) of the argument, as shown below.



The argument may be any floating point value or expression, but must be positive. A negative argument gives a value of 0 for the Square_root function.

For example, the following expression is evaluated as 5 if the user variable Cut_Length has a value of 25.

Square_root(Cut_Length)

Using AxisLink

The AxisLink option for your motion controllers lets you use axes on other motion controllers (also with the AxisLink option) or an ALEC (AxisLink Encoder Converter) as master axes for electronic gearing and cams. This means that axes on different motion controllers can be synchronized, providing real-time coordination for distributed, multi-axis systems in electronic gearing, cam, and lineshaft applications. Sixteen discrete inputs and outputs per motion controller are available to other motion controllers via AxisLink for sequencing and program synchronization.

Important: Be sure to set all motion controllers and ALEC modules, on the AxisLink network, to the same servo update rate.

You can connect up to either of the following:

- Eight modules (motion controllers or ALECs), using only AxisLink, allowing a maximum of 32 coordinated or synchronized axes, if all modules are four-axis motion controllers.
- 16 modules (motion controllers or ALECs), if Extended AxisLink is enabled in the General page of the Configure Control Options dialog box, allowing a maximum of 64 coordinated or synchronized axes, if all modules are four-axis motion controllers.

These modules can be placed as needed along the AxisLink cable, provided the total length of the cable does not exceed 85 feet (25 meters) in normal operation. For 1394 and Compact motion controllers, total cable length can extend up to 410 feet (125 meters) if operating in extended length mode. In any case, the cable length between any two modules must be 1 meter or more. Refer to *Connecting the AxisLink Cable* later in this section.

Virtual Axes

An axis on one motion controller, which is linked to an axis on another motion controller or ALEC via AxisLink, is called a virtual axis. A virtual axis in one motion controller can be linked to any physical or imaginary axis on any other motion controller or ALEC via AxisLink. Thus, virtual axes are functionally identical to extra encoder inputs on the motion controller.

Each motion controller can define up to two virtual axes, but only one of these can be in use at any time. In normal operation, up to four different axes can simultaneously be in use as virtual axes on AxisLink. However, in extended length mode, only two axes can be used simultaneously by AxisLink. See *Connecting the AxisLink Cable.*

AxisLink I/O

In addition to the virtual axes, each motion controller on AxisLink has 16 user-definable discrete inputs and 16 user-definable outputs. Any motion controller on AxisLink can read the AxisLink outputs of up to seven other motion controllers via AxisLink. Thus, each motion controller on AxisLink has a maximum of 112 (112 = 7 x 16) AxisLink Inputs that can be read and used (for program synchronization, etc.). AxisLink I/O response is less than 1 millisecond.

Note: Enabling Extended AxisLink does not increase the maximum number of AxisLink I/O.

Like other discrete I/O in the motion controllers, AxisLink I/O points can be defined singly or in groups of up to 16 I/O for passing numeric values.

Getting Started

Before attempting to use AxisLink functions in your GML Commander diagram, you must connect the AxisLink cable to all motion controllers or ALECs and enable AxisLink via the Control Options definition in GML Commander as explained below. No motion controller needs to be running a program in order to use the AxisLink functions.

Connecting the AxisLink Cable

For information on connecting the AxisLink cable, including an explanation of extended length mode, referenced above, see *AxisLink* information in the *Installation and Hookup* section of the *Installation and Setup* manual for your motion controller. Also, see *Connecting AxisLink* in the *Installation & Hookup* chapter of the *ALEC Installation and Setup* manual (publication 4100-5.3).

Setting the AxisLink Addresses

Set the addresses of AxisLink modules in either of two ways:

If Extended AxisLink is:	Then:
Not enabled	Use the front panel rotary Address switch. Each module must have a unique address between 0 and 7. Do not select positions 8 - F. The order of addresses on the link is not important—only that no module has the same address as any other.
Enabled	Set the addresses of all AxisLink modules in the AxisLink page of the Configure Control Options dialog box. GML Commander ignores the front panel rotary Address switch.

If the motion controller or ALEC is powered when the address is changed, press the RESET switch on the front panel to ensure that the motion controller or ALEC recognizes the new address.

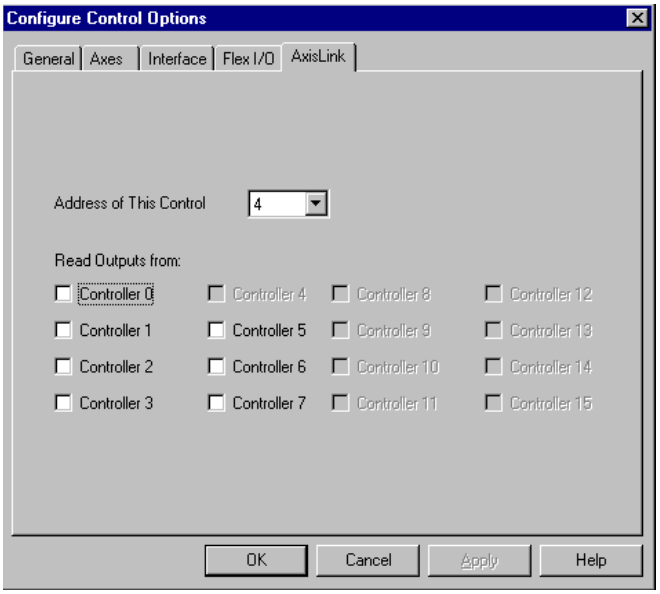
Enabling AxisLink in GML Commander

To enable AxisLink:

1. Select **Configure** from the menu bar. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select **AxisLink** in the *Interfaces* section of the General page of the Configure Control Options dialog box. This selection activates Axis-Link, and displays the AxisLink dialog tab.

Note: If you also select Extended AxisLink, your controller can read output from up to 15 remote motion controllers or ALECs. This Extended AxisLink option is available only for 1394 or Compact controllers using iCODE version 3.5 or higher.

4. Select the **AxisLink** tab. The AxisLink page appears.



5. In the *Address of This Control* field select the address of the controller to which the current GML Commander diagram is downloaded. (The values are from 0 to 7, or from 0 to 15 with Extended AxisLink.)

6. Select up to seven remote controllers from which output can be read.

7. Select **OK**.

In the example shown on the screen, the selection indicates a motion controller with an ADDRESS setting of 4. The addresses of the controllers from which outputs are to be read have not yet been selected, and Extended AxisLink has not been selected (thus, only 7—and not 15—remote controllers are available). See *Defining AxisLink Inputs*.

AxisLink is not finally enabled until the diagram is downloaded to the controller. To enable AxisLink immediately, do one of the following:

- Create a null diagram by connecting the Start block to the End block then download the null diagram to the motion controller.
- Use a Control Setting function block, with Adjust selected, to enable the AxisLink Data Bit.

Once AxisLink is enabled in all modules on the link, the AxisLink status LED on the motion controllers and the AL LED on any ALECs should be green. If any AxisLink LED on any module is red, reset that module using the front panel RESET button. If, after re-setting, any of the LEDs again turns red, check your cabling and termination to ensure that all modules are correctly connected and terminated (if necessary).

See *Connecting AxisLink* in the *Installation and Hookup* section of your motion controller's *Installation and Setup* manual. Also, see *Connecting AxisLink* in the *Installation & Hookup* chapter of the *ALEC Installation and Setup* manual (publication 4100-5.3) for information on connecting and terminating AxisLink cables.

Note: Do not proceed until all AxisLink Status or AL LEDs, on all AxisLink modules, are green.

Configuring Virtual Axes

Virtual axes must be configured in the GML Commander diagram for any motion controllers linked to a physical or imaginary axis on another motion controller or ALEC. Motion controllers whose axes are linked to only other motion controllers do not need to have any virtual axes defined. In addition, virtual axes need not be defined for controllers using only AxisLink I/O.

AxisLink provides two virtual axes for use in each GML Commander diagram. However, only one of these can be in use at any time. The second virtual axis is provided to facilitate switching between two different master axes on the fly. If the motion controller does not need to switch master axes on the fly, only one of the virtual axes needs to be defined.

To configure a virtual axis:

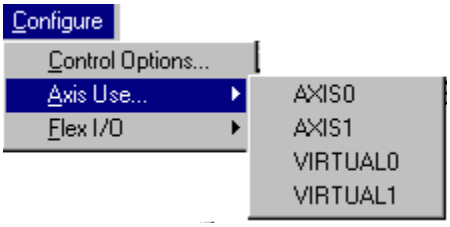
1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select **AxisLink** in the *Interfaces* section of the General page of the Configure Control Options dialog box. This selection activates AxisLink, and displays the AxisLink dialog tab.

   Note: If you also select Extended AxisLink, an additional eight AxisLink nodes (i.e., motion controllers on the link) become available.

4. In the General page of the Configure Control Options dialog box, select the Axes tab.
5. In the Axes page of the Configure Control Options dialog box, select one (or both) virtual axis. This enables the selected axis (or axes).

   Note: You can also rename a virtual axis in this dialog box.

6. From the menu bar, select **Configure**. The Configure menu appears.
7. Select **Axis Use**. The Configure Axis Use dialog box appears.
8. Select a virtual axis from the Axis Use menu shown below:



The Configure Axis Use dialog box appears.

9. In the General page of the Configure Axis Use dialog box, make the following entries:

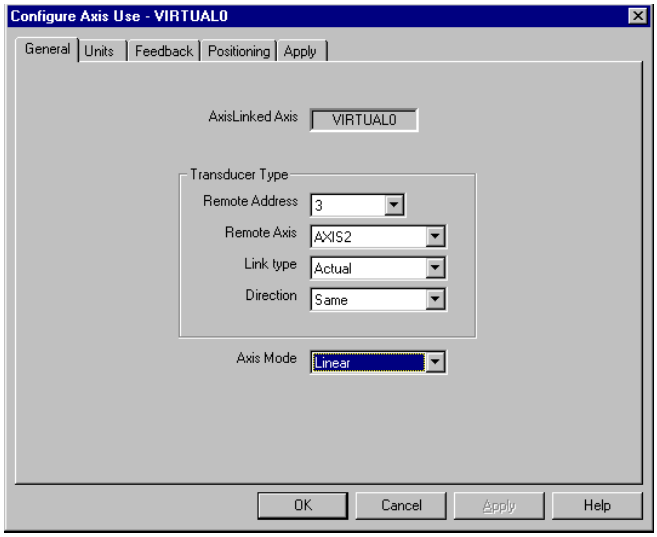
Field	Description
Remote Address	Select the remote address of the controller or ALEC to which this virtual axis is to be linked. Note: For controllers: read this address from the front panel ADDRESS selector switch of the appropriate module (if Extended AxisLink is not enabled), or from the AxisLink page of the Configure Control Options dialog box (if Extended AxisLink is enabled).
Remote Axis	Select a physical axis on the distant controller to which this controller is linked.

10. Complete virtual axis configuration by downloading the configuration settings to the controller. To do this, do one of the following:

- Create a null diagram by connecting the Start block to the End block and download the diagram to the motion controller.
- From the Apply page of the Configure Axis Use dialog box, select **Download**.

Example of a virtual axis configuration

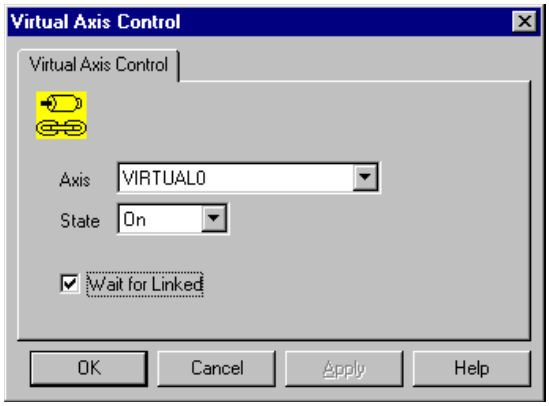
The axis configuration, shown below, links virtual axis 0 in this motion controller to physical Axis 2 of the motion controller whose address is 3 via AxisLink.



Like a physical axis, set each AxisLink virtual axis' Mode (rotary or linear), Units and Display Formats, Conversion Constant, Unwind Constant (rotary axes only) and Home Position. However, these settings for the virtual axis need not be identical to those for the physical axis to which the virtual axis is linked. See Axis Use dialog box for information on making these selections.

Using Virtual Axes

Once a virtual axis has been configured and set-up, it must be enabled via a Virtual Axis Control block before it can be used as a master axis for electronic gearing or position-lock cams. For example, a Virtual Axis Control block with the following settings enables virtual axis 0 (and disables virtual axis 1, if it was previously enabled) and pauses program flow until the link is established.



See *AxisLink Blocks* in the *Functional Blocks* chapter of this manual for more information on the Virtual Axis Control block.

After enabling a virtual axis, the variable Axislink_status = 1 for that virtual axis. The virtual axis can be tested, while moving the linked motion controller's physical axis, by reading its actual position using General Watch.

Because AxisLink virtual axes are functionally equivalent to physical Master Only axes, they can be used in place of a physical axis in the following blocks:

- Redefine Position
- Home Axis
- Gear Axes (Master axis only)
- Position Lock Cam (Master axis only)
- On Axis
- Output Cam
- Watch Control
- On Watch
- Show Axis Position

- Show Axis Status
- Control Setting

See the descriptions of the above blocks in the *Function Blocks* chapter of this manual for more information on using virtual axes with each of them.

ATTENTION: Ensure that all motion controllers and ALEC(s) on AxisLink are using the same Servo Update Rate.

When a virtual axis in one controller is AxisLinked to the command position of an axis in a controller using a different servo update rate, the speed of the virtual axis is different from the speed of the corresponding physical or imaginary axis due to the different servo update rates. If the motion controller in which the virtual axis is defined has a faster servo update rate than the controller to which the virtual axis is linked, the virtual axis appears to be going faster than the corresponding physical or imaginary axis and vice versa. For best results, ensure that all motion controllers connected via AxisLink are using the same servo update rate. The Servo Update Rate (parameter D47) can be checked using the Control Setting block in GML Commander by selecting the Show check box in the *Type* area, and selecting Servo Update Rate in the Data Parameter list. See the *Control Settings* block in the *Function Blocks* chapter of this manual for more information.

Homing Virtual Axes

Because the virtual axis has no physical dimensions, only passive homing can be performed on the virtual axis. You can use the Home Axis block to passively home a virtual axis. When a virtual axis is passively homed, the virtual axis' Actual_Position Axis System Variable changes, and (upon completion of passive homing) reflects the virtual axis' new home position. The position of the physical axis on the motion controller (or ALEC), to which the virtual axis is linked, is not affected. Thus, the home position value (entered in the motion controller's machine setup menu) need not be the same for both the virtual axis and the physical axis to which it is linked.

Passively homing the virtual axis—or any axis—causes no motion. The passive homing operation is complete when the physical axis, to which the virtual axis is linked, is caused to move past its encoder marker pulse. See *Home Axis* in the *Function Blocks* chapter of this manual for more information on passive homing. The homing status of the virtual axis can be checked using either its Homing_status variable, or an On Axis block, as is the case with a physical axis.

During passive homing, the Marker_Distance variable for the virtual axis is updated. The value of the Marker_Distance variable is equal to the position of the axis when the marker occurred. The numerical change in the actual position of the virtual axis caused by the passive homing operation can thus be calculated as the value of the Marker_Distance variable minus the axis' Home Position, after passive homing is completed. The Passively Home Axis and Calculate Position Change module included in the GML Commander Module Library on your GML Commander disk implements this calculation for Axis 0. See *Marker Distance* in the *Expression Builder* chapter for more information on this module.

A virtual axis can also be homed using a Redefine Position block to directly set the actual position of the axis to the value that you want. See *Redefine Position* in the *Function Blocks* chapter of this manual for more information.

Registration on Virtual Axes

The Watch Control block can be used to set up a registration event for a virtual axis. When a registration event occurs on a virtual axis, the Registration_Position variable of the physical axis on the motion controller to which the virtual axis is linked is not affected—only the virtual axis' Registration_Position is changed.

However, the registration input of the physical axis to which the virtual axis is linked must be activated or deactivated (as determined by the Watch Control dialog box settings) to cause the registration event on the virtual axis. See *Watch Control* in the *Function Blocks* chapter of this manual for more information on registration. The registration status of the virtual axis can be checked using either its Registration_status variable, or an On Watch function block.

Virtual Axis System Variables

The following table shows the system variables that can be used with virtual axes. See the *System Variables* chapter of this manual for specific information on each of these variables.

GML Commander Virtual Axis System Variables

Type	Variable	Units
Motion	Actual_Position Strobed_Position Registration_Position Marker_Distance Average_Velocity	Axis Position Units Axis Position Units Axis Position Units Axis Position Units Axis Position Units / Sec
Fault	Axis_Fault AxisLink_timeout AxisLink_failed	Integer value: 0, 6, or 7 0 (False) or 1 (True) 0 (False) or 1 (True)
Status	Axis_status Homing_status Registration_status AxisLink_status	Integer value: 6, 12 - 14 0 (False) or 1 (True) 0 (False) or 1 (True) 0 (False) or 1 (True)

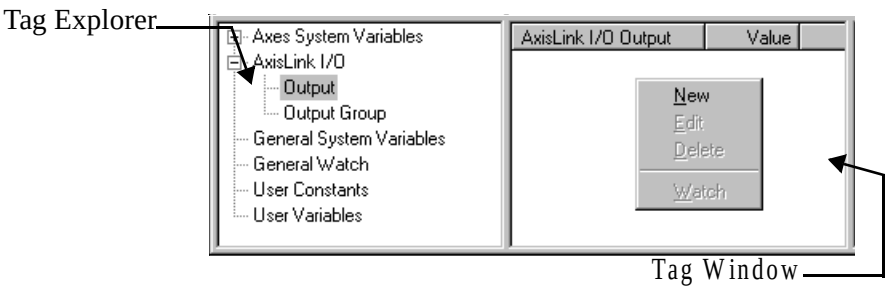
Using AxisLink I/O

Each motion controller on AxisLink has 16 user-definable discrete outputs. These can be read as AxisLink inputs by any other motion controller on AxisLink, and can be used for program synchronization, etc. Thus, each motion controller on AxisLink has a maximum total of 112 (112 = 7 x 16) AxisLink inputs in addition to its own 16 outputs. Like other discrete I/O in the motion controllers, AxisLink I/O can be defined singly or in groups of up to 16 I/O.

AxisLink Outputs

Like other discrete I/O in the motion controllers, the 16 discrete AxisLink outputs must be defined before being used in the GML Commander diagram. You can define AxisLink outputs individually, or in groups of up to 16 outputs, as follows.

1. In the Tag Explorer (see below), highlight either **Output** or **Output Group**.
2. To define a new output, in the Tag Window, do one of the following:
   - Double-click the left mouse button.
   - Single-click the right mouse button.
3. In the pop-up menu, select **New** (see below)



4. In the AxisLink I/O Output (or Output Group) dialog box, complete the output name, address and (if applicable) group I/O options settings. See *Online Help* for further instructions.

AxisLink Inputs

I/O connections to other motion controllers must be configured before any AxisLink inputs from these other linked motion controllers can be defined and used in the GML Commander diagram.

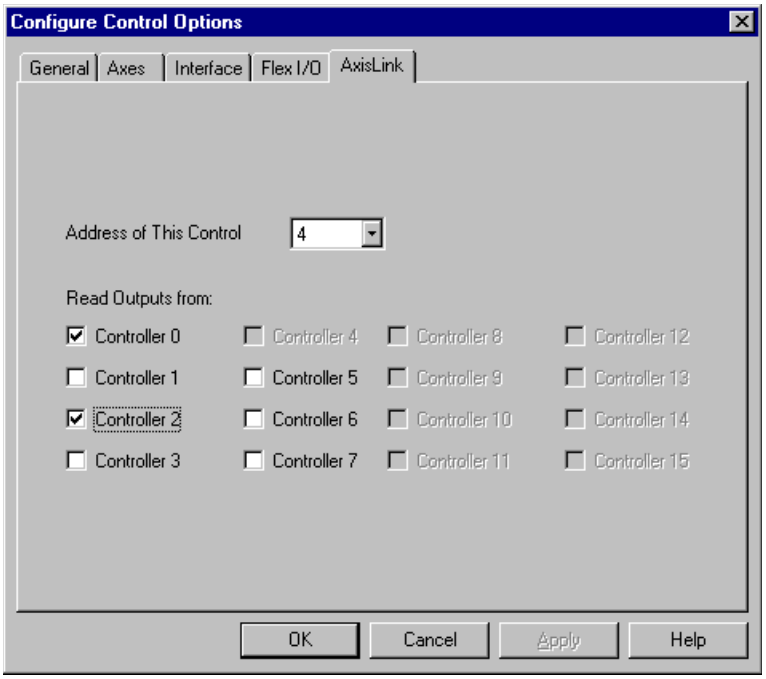
Configuring AxisLink Inputs

To configure the AxisLink I/O input connections:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options menu appears.
3. In the *Interfaces* section of the General page, select **AxisLink**. The AxisLink tab appears.

   Note: If you also select Extended AxisLink, your controller can read output from up to 16 remote motion controllers or ALECs. This Extended AxisLink option is available only for 1394 or Compact controllers using iCODE version 3.5 or higher.

4. Select the **AxisLink** tab.
5. Select the **Address of This Control** to which the current GML Commander diagram is downloaded.
6. Select up to seven remote controllers from which output can be read. In the following example, the current diagram is downloaded to the motion controller with the defined Address 4 which, in turn, reads as AxisLink inputs the AxisLink outputs from those motion controllers with addresses defined as 0 and 2.



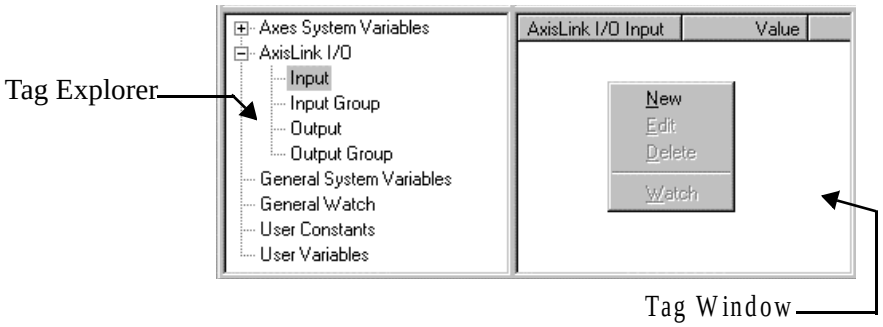
Note: In the above example, controller address number 4—the address of the controller receiving input—is not available. Addresses 8 through 15 are enabled only by selecting Extended AxisLink in the General page of the Configure Control Options dialog box.

Defining AxisLink Inputs

After the AxisLink I/O Connections have been configured, the next step is to define the AxisLink inputs that are used in the GML Commander diagram.

1. In the Tag Explorer, under AxisLink I/O, highlight either **Input** or **Input Group**.
2. To define a new input, in the Tag Window, do one of the following:
   - Double-click the left mouse button.
   - Single-click the right mouse button.

A menu appears.



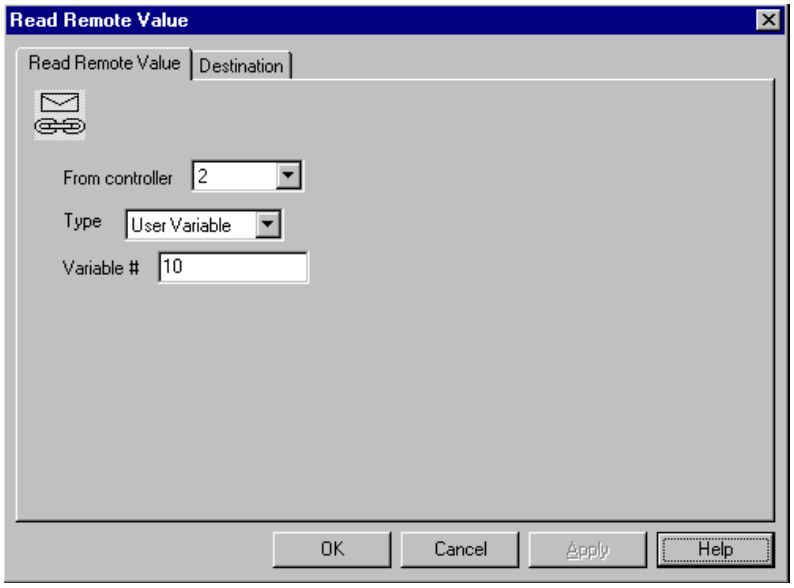
3. Select **New**. The AxisLink I/O Input (or Input Group) dialog box appears.

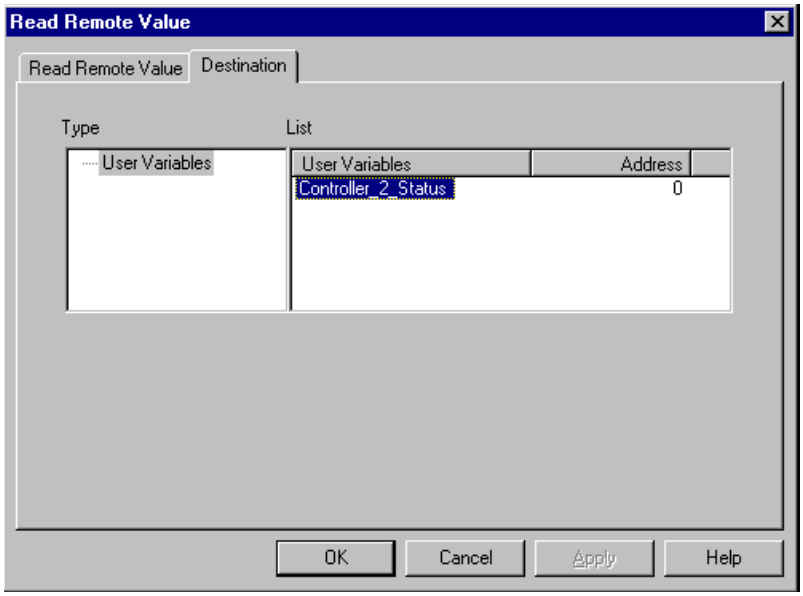
4. Complete the input name, address and (if applicable) group I/O options settings. See *Online Help* for more information on defining AxisLink inputs and input groups.

Reading Values via AxisLink

Each motion controller on AxisLink can request the current value of a user variable, data parameter, or data bit from another motion controller on the link using the Read Remote Value block. GML Commander stores the returned value in a user variable in the controller that executed the Read Remote Value block. Usually, values are returned in 3 to 5 milliseconds, but can take as long as 20 milliseconds. Both motion controllers must have the AxisLink option both installed and enabled and be connected to the same AxisLink cable.

For example, suppose this motion controller needs to know the status of another motion controller (with an address of 2) on AxisLink. Motion controller 2 has a user variable (User Variable 10) defined that contains a numeric value indicating its status. The following Read Remote Value block in the GML Commander diagram for this motion controller, is then used to read the status variable from controller 2 and store the value in the local user variable Controller_2_Status. See *AxisLink Block* in the *Function Blocks* chapter of this manual for more information on the Read Remote value block.





It is not possible to directly write a variable or parameter value to another controller via AxisLink. However, a discrete AxisLink output on this motion controller can be used to tell another motion controller to request a value using a Read Remote Value block, thus accomplishing the same thing.

Handling Faults

If the application program (generated from the GML Commander diagram and downloaded to the motion controller) is not running, each motion controller continually attempts to establish all configured virtual axis and AxisLink I/O links. This guarantees that soon after the last motion controller powers up, all configured links are working properly. When the application program is running, the controller does not attempt to re-establish any failing links—this must be done by the fault handling routine in the program.

In most applications, AxisLink faults should be handled as part of a global fault handling task or routine in the diagram. See *End Program* in the *Function Blocks* chapter of this manual for information on the recommended global fault handler. Since an AxisLink fault in one motion controller can also cause AxisLink faults in other motion controllers in the chain, the fault recovery routine must consider the application programs running in other motion controllers. The simplest solution is to restart all the application programs via a hardware or software reset.

There are two types of AxisLink faults:

- Virtual axis faults.
- General faults associated with AxisLink I/O and the reading of data values via AxisLink.

Virtual Axis Faults

When a fault occurs on a virtual axis, the Global_fault variable has a value of 9, and the message “AxisLink Virtual Axis Fault” appears in the General System Variable Watch window. A virtual axis fault (Global_fault = 9) indicates that an AxisLink virtual axis connection has failed or could not be established (timeout).

If a virtual axis link that had been operating properly fails to provide any new information for at least 4 servo update intervals, the connection times out. When this timeout occurs, the AxisLink_timeout = 1, Axis_fault = 7, Axis_status = 14, and Global_fault = 9. In addition, the AxisLink LED on the front panel flashes red.

The typical causes of an AxisLink timeout are:

- The AxisLink cable has broken or become disconnected.
- The motion controller or ALEC, containing the physical axis to which the virtual axis is linked, has failed or has disabled AxisLink.

If a virtual axis connection could not be established by a Virtual Axis Control block (the axis, motion controller, or ALEC could not be found), the link is said to have failed. When an attempted connection fails, AxisLink_failed = 1, Axis_fault = 6, Axis_status = 13, and Global_fault = 9. In addition, the AxisLink LED on the front panel flashes green. The typical causes of an AxisLink failure are:

- The controller address specified in the Virtual Axis Control block does not match any of the modules (motion controllers or ALECs) on the link.
- The module containing the physical axis to which the virtual axis is linked has failed, has disabled AxisLink, or has not finished powering up.
- The physical axis on the controller specified in the Virtual Axis Control block does not exist (axes 2 and 3 on a two-axis controller, for instance) or is disabled in the Axis Use definition in that controller.
- The AxisLink cable is broken or disconnected.

After identifying and correcting the cause of the problem, clear the virtual axis fault with a Reset AxisLink Fault block or by setting the AxisLink_timeout or AxisLink_failed variable for the virtual axis to 0 using an Equation block. Then re-enable the virtual axis using a Virtual Axis Control block.

AxisLink General Faults

When an AxisLink general fault (an AxisLink fault not directly associated with a virtual axis) occurs, the Global_fault variable has a value of 8, and the message “AxisLink General Fault” appears in the global fault field in the Terminal window (and in the Runtime Display, if enabled). An AxisLink general fault (Global_fault = 8) is second in priority only to an AxisLink virtual axis fault.

An AxisLink general fault is caused by:

- By an AxisLink timeout while attempting to access another motion controller’s AxisLink outputs or data.

- By communication errors on the link.
- If AxisLink on this controller is offline (not operating).

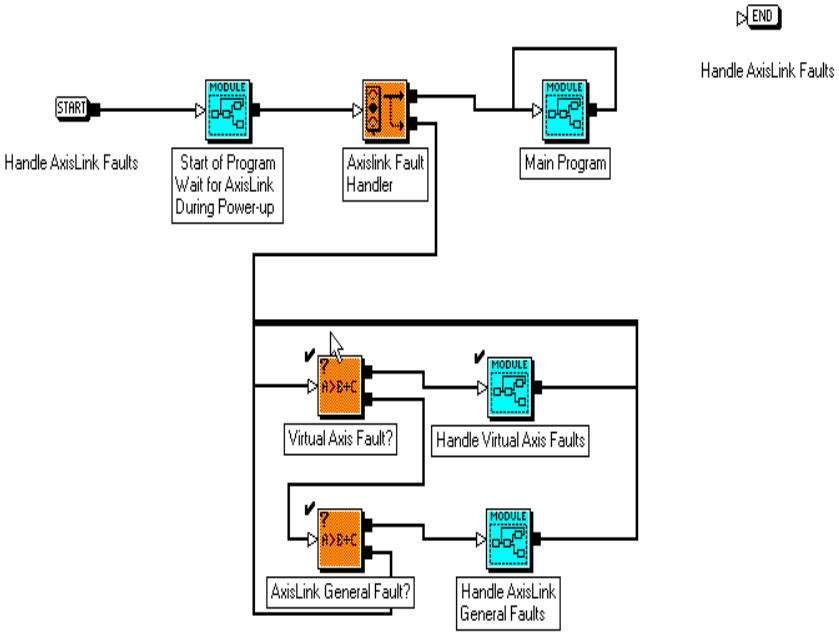
When one of these faults occur, AxisLink_general_fault = 1 and Global_fault = 8 if no virtual axis faults are active.

The specific cause of an AxisLink general fault can be determined by the value of the AxisLink_fault_code variable. See the *System Variables* i chapter in this manual for a description of the AxisLink_fault_code values and the causes of each condition. After correcting the cause of the problem, clear the AxisLink general fault with a Reset AxisLink Fault block or by setting the AxisLink_general_fault variable to 0 using an Equation block.

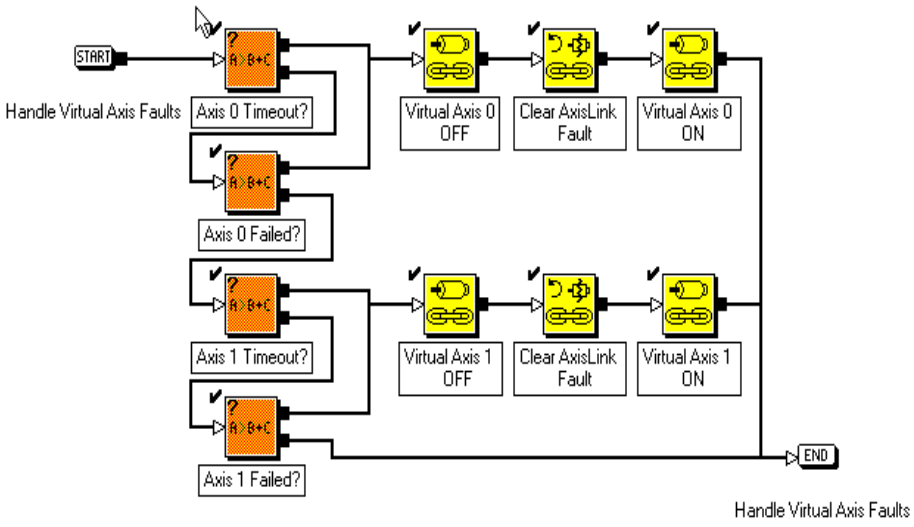
AxisLink Fault Handling Module

The Handle AxisLink Faults module, included in the module library on the GML Commander disk, implements complete fault handling for all AxisLink faults as discussed above. Use this module in the global fault handling routine for all motion controllers on AxisLink. This lets each motion controller recover from both its own AxisLink faults, and those caused by faults in other modules on the link.

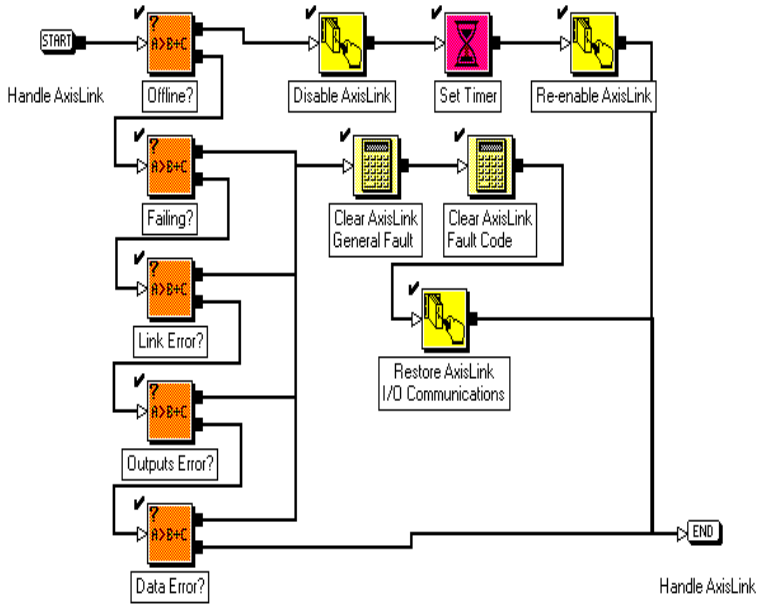
The Handle AxisLink Faults module first determines whether the fault is a virtual axis fault or an AxisLink general fault by checking the value of Global_fault . A separate module handles each of these two types of fault, as shown below.



The Handle Virtual Axis Faults module checks for a timeout or failure on both virtual axes. The faulty axis is disabled, the fault is cleared, and the virtual axis re-enabled as shown below.

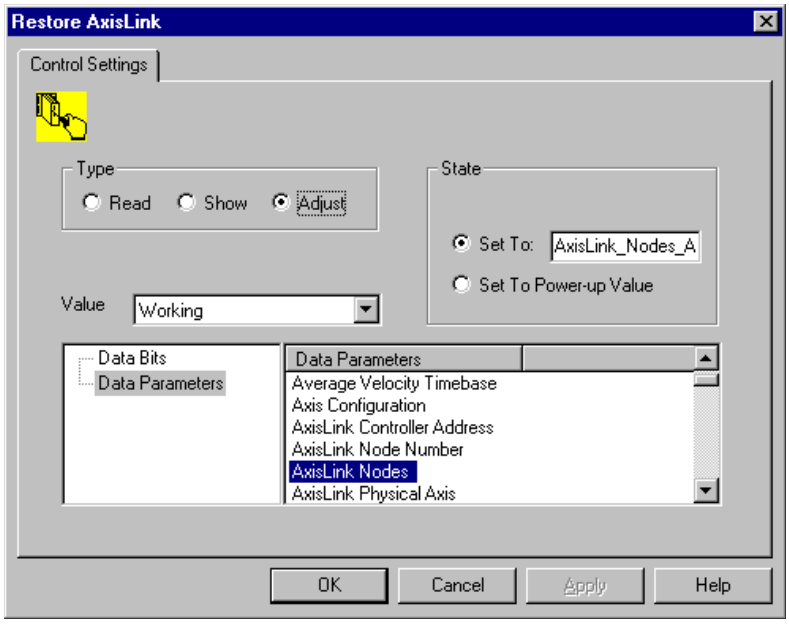


The Handle AxisLink General Faults module checks for all possible values of the AxisLink_fault_code variable and takes the appropriate action, as shown below.



If the link is offline, AxisLink is disabled and then re-enabled—this is the only way to clear the offline condition. As long as the link is not offline, AxisLink data and I/O communication with other controllers is temporarily stopped, the fault variables are cleared, and data and I/O communication with other controllers restored.

To restart AxisLink I/O communication without disrupting any virtual axes that could be operating, after an AxisLink General Fault has occurred, the data parameter AxisLink_Nodes, parameter (D63)—which stores the list of other controllers from which this controller is reading outputs via AxisLink—is restored with a Control Settings block. There, the working value of the AxisLink_Nodes data parameter is adjusted by setting it to the value of the user variable AxisLink_Nodes_Addresses, as shown on the following screen.



Note: In the block, above, the AxisLink_Nodes_Addresses user variable was declared for use by another Control Settings block (found in the Start of Program GML Commander module, included in the GML Commander module library). The AxisLink_Nodes_Addresses user variable was input as the Set To value using the Expression Builder.

For more information on user variables, see the *User Variables* chapter in this manual.

Performance Considerations

Enabling AxisLink increases the CPU utilization of the motion controller due to the communication overhead of the link. Enabling a virtual axis again increases the CPU utilization because—in effect—it adds another Master Only axis to the controller. Four-axis motion controllers should use a 500 Hz servo update rate if they are also using an AxisLink virtual axis.

On the other hand, the CPU utilization of the motion controller to which a virtual axis is linked is not increased by having one of its axes linked to by another controller or multiple controllers. Once an AxisLink connection is established between two modules (motion controllers or ALECs), reading AxisLink inputs, setting AxisLink outputs, or reading values over AxisLink does not affect the CPU utilization of either controller.

Using the RIO Adapter Option

The Remote I/O option provides RIO Adapter functionality for the 1394 GMC/GMC Turbo, Compact, Integrated, and Basic motion controllers. For IMC-S/20x-R and IMC/S21x-R models, the Remote I/O option also provides Scanner functionality. Either way, this option allows the motion controller to communicate directly with an Allen-Bradley PLC or host computer via RIO.

When configured as an Adapter, the motion controller can communicate directly with an A-B PLC using both discrete and block transfers.

When configured as a scanner, IMC-S/20x-R and IMC/S21x-R model motion controllers can:

- Communicate directly to an operator interface (such as Panelview).
- Communicate directly to another IMC-S/20x-R and/or IMC/s21x-R model motion controller.
- Distribute discrete I/O (such as 1791 block I/O or 1794 Flex I/O).

When configured as a scanner, the motion controller can communicate with up to one full rack of discrete I/O at a single address.

This chapter focuses on the RIO Adapter option.

Using the RIO adapter, certain aspects of the motion controller can be controlled and monitored from the PLC. Both discrete and block transfers are available.

Discrete Transfers

Discrete transfers are used for the exchange of real-time data via discrete inputs and outputs. To both the motion controller and the PLC, discrete transfer I/O looks like physical discrete I/O and is programmed or accessed in the same manner.

When configured as an adapter, both dedicated (pre-defined) and user-defined discrete I/O are available. It provides 12 dedicated discrete inputs for direct control of certain manual functions and 12 dedicated discrete outputs for basic status reporting to the PLC. Up to 100 user-defined inputs and 100 user-defined outputs are also available. These I/O are transferred once each RIO scan.

Block Transfers

Block transfers are used to transfer parameter values, recipes, status, and other non time-critical data such as axis position, velocity, etc. for display purposes. Up to 64 words (128 bytes) of data can be transferred from the motion controller to the PLC or from the PLC to the motion controller in each block transfer. In addition, data can be transferred in one of four different numeric formats as required by the RIO scanner or other devices on the RIO link. Note that compared to direct transfers, there can be a relatively long time between block transfers depending on the scan rate of the PLC and the number of devices on the RIO link.

Block transfers allow the PLC to read and write variable, data parameter, data bit, and cam table values directly in binary format to the motion controller. The PLC stores these values in data files.

Only the RIO scanner (in the PLC or host computer) can initiate block transfers. However, the motion controller can request a block transfer from the PLC using a user-defined discrete output.

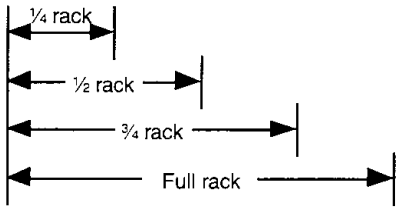
Important: Do not create a PLC program in which a Block Transfer Read and a Block Transfer Write are read in the same I/O scan. A RIO error occurs.

See *Using Block Transfers* in this chapter for more information on working with block transfers.

RIO Addressing

To the PLC, the motion controller appears as 1/4, 1/2, 3/4, or a full remote rack on the RIO link depending on the setting of the Rack Size parameter in the RIO page of the Configure Control Options dialog box, as explained below.

IMC S Class RIO Adapter



Block Transfers (PLC); Reserved (SLC)	Dedicated Discrete I/O						
		4	20	36	52	68	84
		5	21	37	53	69	85
		6	22	38	54	70	86
		7	23	39	55	71	87
		8	24	40	56	72	88
		9	25	41	57	73	89
		10	26	42	58	74	90
		11	27	43	59	75	91
		12	28	44	60	76	92
		13	29	45	61	77	93
		14	30	46	62	78	94
		15	31	47	63	79	95
	0	16	32	48	64	80	96
	1	17	33	49	65	81	97
	2	18	34	50	66	82	98
	3	19	35	51	67	83	99

In the figure above, the numbers 0 – 99 correspond to the RIO Input and Output address defined in GML Commander. As implied in the figure above and shown in the following table, the selected rack size determines the total number of user-defined discrete I/O.

Remote I/O Adapter
Number of User-Defined Discrete I/O Bits

Rack Width	Inputs	Outputs
¼	4	4
½	36	36

Remote I/O Adapter Number of User-Defined Discrete I/O Bits		
Rack Width	**Inputs**	**Outputs**
¾	68	68
Full	100	100

In general, set the rack size to the smallest value that provides enough user-defined discrete I/O for your application.

The Starting Group parameter in the RIO page of the Configure Control Options dialog box determines where in the rack the motion controller RIO adapter appears to the RIO scanner.

Starting Group	I/O Group								SLC Bit Number	PLC Bit Number
0:	0	1	2	3	4	5	6	7		
2:	2	3	4	5	6	7	N/A	N/A		
4:	4	5	6	7	N/A	N/A	N/A	N/A		
6:	6	7	N/A	N/A	N/A	N/A	N/A	N/A		
Block Transfers (PLC); Reserved (SLC)	Dedicated Discrete I/O									
		4	20	36	52	68	84		0	0
		5	21	37	53	69	85		1	1
		6	22	38	54	70	86		2	2
		7	23	39	55	71	87		3	3
		8	24	40	56	72	88		4	4
		9	25	41	57	73	89		5	5
		10	26	42	58	74	90		6	6
		11	27	43	59	75	91		7	7
		12	28	44	60	76	92		8	10
		13	29	45	61	77	93		9	11
		14	30	46	62	78	94		10	12
		15	31	47	63	79	95		11	13
	0	16	32	48	64	80	96		12	14
	1	17	33	49	65	81	97		13	15
	2	18	34	50	66	82	98		14	16
	3	19	35	51	67	83	99		15	17

IMC S Class RIO Adapter

Important: SLC processors should not use the I/O Group reserved for block transfers in the motion controller RIO Adapter.

For example, with a rack size of 1/2 and a starting group of 4, block transfers appear in I/O Group 4, the dedicated I/O and the first four user-defined I/O in I/O Group 5, user-defined I/O 4 – 19 in I/O Group 6, and user-defined I/O 20 – 35 in I/O Group 7. As indicated by N/A, user-defined discrete I/O 36 – 99 are not available with this addressing configuration. In addition, SLC processors should not use the I/O Group reserved for block transfers in the motion controller RIO Adapter.

The motion controller RIO adapter discrete inputs and outputs appear in the output or input image table (respectively) of the SLC or PLC as shown in the columns on the right. Continuing our example, motion controller RIO adapter discrete output 17 appears in the input image table of an SLC as I:RA6/13 or of a PLC as I:RA6/15, where RA is the Rack Address specified in the RIO page of the Configure Control Options dialog box.

Getting Started

Before attempting to use RIO functions in your GML Commander diagram, you must connect and set up the RIO adapter in the motion controller and enable it using the Configure Control Options dialog box in GML Commander as explained below. Because it is a slave on the RIO link, the motion controller need not be running a program to use the RIO functions. However, the PLC or host computer containing the RIO scanner (the master on the RIO link) must be running a program.

Connecting the RIO Cable

See *Connecting Remote I/O* in the *Installation and Hookup* section of the *Installation and Setup* manual for your motion controller to connect the RIO cable to the motion controller.

Configuring the RIO Adapter

Configure the RIO adapter—including baud rate, rack size, and rack address—using either the motion controller's built-in setup menus or the RIO page of the Configure Control Options dialog box in GML Commander. See *RIO Addressing* in this chapter for more information on configuring the RIO adapter for your application.

Enabling RIO in GML Commander

To enable RIO:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. In the General page of the Configure Control Options dialog box, select **RIO**.
4. Select **Adapter** to allow the RIO adapter features of GML Commander to be used in your diagram.
5. See *Online Help* for instructions on completing each of the Configure Control Options pages.

Using the Dedicated Discrete Inputs

RIO Adapter Dedicated Discrete Inputs
I/O Group: Starting Group + 1

Output		
SLC	**PLC**	**Function**
0	0	Select Axis 0 for Jog or Home
1	1	Select Axis 1 for Jog or Home
2	2	Select Axis 2 for Jog or Home
3	3	Select Axis 3 for Jog or Home
4	4	Run application Program
5	5	Stop Application Program
6	6	Pause/Resume Application Program
7	7	Kill Control
8	10	Home Selected Axes
9	11	Jog Selected Axes in a Positive Direction
10	12	Jog Selected Axes in a Negative Direction
11	13	Reserved (Do Not use)

These inputs allow the PLC to run, stop, pause, or resume the application program (generated from the GML Commander diagram and downloaded to the motion controller); directly jog or home one or more axes in the motion controller when the program is not running; or send a kill control signal to the motion controller.

Homing Axes

To directly home an axis from the PLC, select the axis by setting the appropriate axis select bit (0 – 3) and the home selected axes bit (SLC output 8, PLC output 10). To home more than one axis simultaneously, set more than one axis select bit. Axes can be homed using this bit only while there is no application program running in the motion controller. This bit has no effect if it is set while the application program is running.

Note: Do not set the Home Selected Axes bit while the application program in the motion controller is running.

The home selected axes bit turns feedback ON for the selected axes and initiates homing using the power up homing configuration set for each selected axis in the Homing page of the Configure Axis Use dialog box. The power up homing configuration can be active, passive, or absolute, depending on the requirements of the application.

The home selected axes input is edge-sensitive, not level-sensitive. The motion controller must see a transition of this bit from 0 to 1 to initiate homing. After homing has been initiated, the select axis bits and the home selected axes bit are cleared.

The recommended ladder logic for homing axes is shown below. Only axes 0 and 1 are shown, but axes 2 and 3 are handled in the same manner. The Rack Address in the motion controller is set to 5 and the Starting Group parameter to 0. Thus the dedicated discrete RIO bits appear in I/O group 1 of rack 5 (I/O:051).

```
Program Listing Report           PLC-5/30

Rung 2:1
| Home                                                          Axis 0     |
| Axis 0                                                        Selected   |
|   I:002                                                       O:051      |
+----] [-----------------------------------------------------( )-----+
|      01                                                          00      |
Rung 2:2
| Home                                                          Axis 1     |
| Axis 1                                                        Selected   |
|   I:002                                                       O:051      |
+----] [-----------------------------------------------------( )-----+
|      02                                                          01      |
Rung 2:3
|                                                               Home       |
|  Axis 0                                                       Selected   |
|  Selected                                                     Axes       |
|    O:051                                                      O:051      |
+-+---] [----+-------------------------------------------------( )-----+
| |     00   |                                                       10      |
| |Axis 1    |                                                               |
| |Selected  |                                                               |
| |  O:051   |                                                               |
| +---] [----+                                                               |
|       01                                                                   |
```

ATTENTION: Do not use the PLC ladder logic shown here for homing axes in the same PLC program with the ladder logic shown later for jogging axes or improper operation can result.

While homing is in progress, (provided there are no faults active on the axis) both the axis locked and axis fault dedicated discrete RIO outputs from the motion controller for the selected axes are cleared. When homing has finished successfully and the position error of the axis is less than the position lock tolerance, the axis locked dedicated discrete RIO outputs from the motion controller for the selected axes are set. If a fault occurs during homing, the axis fault outputs for the selected axes are set, but the axis locked outputs are cleared. See *Dedicated Discrete Outputs* later in this section for more information on the axis locked and axis fault bits.

The simple PLC ladder logic shown here for homing axes should not be used in the same PLC program with the ladder logic shown later for jogging axes, or improper operation can result. To provide homing and jogging together via the PLC, the home and jog functions must be interlocked with each other. This ensures that homing one axis does not inadvertently jog (or stop jogging) another axis, or vice-versa, because both functions use the same axis select bits.

Jogging Axes

To directly jog an axis from the PLC:

1. Select the axis by setting the appropriate axis select bit (0 – 3).
2. Specify the jogging direction by selecting the jog selected axes bit for the direction:
   - SLC output 9 or PLC output 11 for the positive direction;
   - SLC output 10 or PLC output 12 for the negative direction

   Note: Do not set a Jog Selected Axes bit while the application program in the motion controller is running.

To jog more than one axis simultaneously, set more than one axis select bit. Axes can be jogged only by using these bits while no application program is running in the motion controller. These bits have no effect if set while the application program is running.

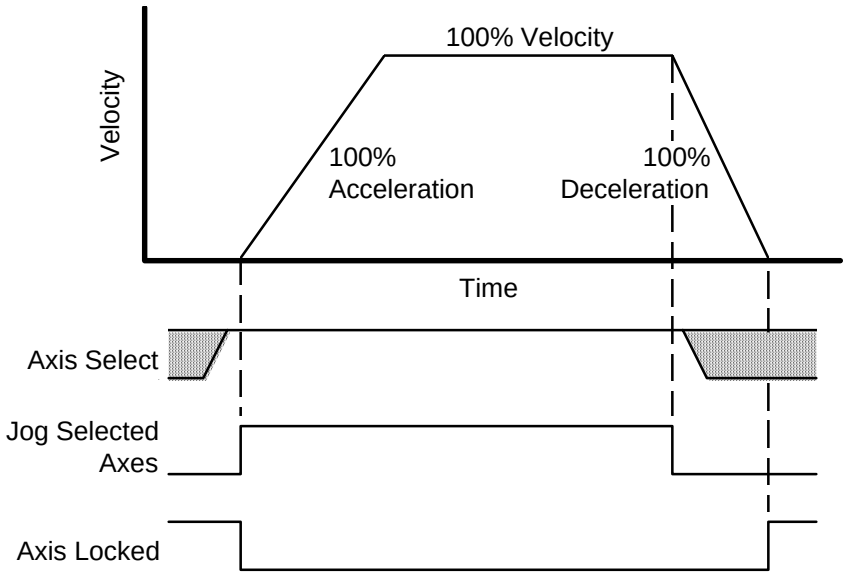
The jog selected axes inputs continuously move the selected axis (or axes) in the specified direction at 100% of maximum speed, and accelerates and decelerates at 100% of that axis' maximum acceleration and maximum deceleration values. The Acceleration, Deceleration and Velocity values are set to the maximum values in the Dynamic page of the Configure Axis Use dialog box for each axis, unless expressly changed by either:

- Motion Settings block (Set Maximum Speed, Set Maximum Accel, or Set Maximum Decel) in a previously running program.
- Block transfer write to the appropriate axis data parameters.

The power up jog profile (Trapezoidal, S-Curve, or Parabolic), set in the Positioning page of the Configure Axis Use dialog box, governs this motion. See *Online Help* for more information on setting the jog parameters.

ATTENTION: The Axis Select bits must remain set until the active Jog Selected Axes bit is cleared.

The jog selected axes inputs are edge-sensitive, not level-sensitive. This means that the motion controller must see a transition of these bits from 0 to 1 to initiate jogging and a transition from 1 to 0 to stop the jog. The select axis bits must remain set until the appropriate jog selected axes bit is cleared to stop the jog, as shown below, otherwise unexpected operation can result.



While a jog is in progress, both the axis locked and axis fault dedicated discrete RIO outputs from the motion controller for the selected axes are cleared, provided there are no faults active on the axis. When the jog has finished successfully and the position error of the axis is less than the position lock tolerance, the axis locked dedicated discrete RIO outputs from the motion controller for the selected axes are set. If a fault occurs during the jog, the axis fault outputs for the selected axes are set, and the axis locked outputs are cleared. See *Dedicated Discrete Outputs* later in this section for more information on the axis locked and axis fault bits.

The recommended ladder logic for jogging each of two axes from individual positive and negative jog inputs to the PLC is shown in the following exhibit. The rack address in the motion controller is set to 5 and the starting group parameter to 0. Thus the dedicated discrete RIO bits appear in I/O group 1 of rack 5 (`I/O:051`).

```
Program Listing Report                PLC-5/30
Rung 2:1
|    Jog 0      Axis 1                                          Axis 0   |
|    Positive   Selected                                        Selected |
|    I:002      O:051                                           O:051    |
+-+---] [----+---]/[----+-----------------------------( )-----+
| |     00   |     01   |                                        00    |
| |  Jog 0   |          |                                              |
| |  Negative|          |                                              |
| |  I:002   |          |                                              |
| +---] [----+       |                                               |
| |     01           |                                               |
| |  Axis 0     Axis 0  |                                              |
| |  Selected   Locked  |                                              |
| |  O:051      I:051   |                                              |
| +---] [--------]/[----+                                             |
|       00         00                                               |
Rung 2:2
|    Jog 1      Axis 0                                          Axis 1   |
|    Positive   Selected                                        Selected |
|    I:002      O:051                                           O:051    |
+-+---] [----+---]/[----+-----------------------------( )-----+
| |     02   |     00   |                                        01    |
| |  Jog 1   |          |                                              |
| |  Negative|          |                                              |
| |  I:002   |          |                                              |
| +---] [----+      |                                                |
| |     03           |                                               |
| |  Axis 1     Axis 1  |                                              |
| |  Selected   Locked  |                                              |
| |  O:051      I:051   |                                              |
| +---] [--------]/[----+                                             |
|       01         01                                               |
Rung 2:3
|    Jog 0                Axis 0                                Jog Axes |
|    Positive   Jog -     Fault                                 Positive |
|    I:002      O:051     I:051                                 O:051    |
+-+---] [--------]/[--------]/[----+------------------( )-----+
| |     00         12         04   |                              11    |
| |  Jog 1                  Axis 1  |                                    |
| |  Positive   Jog -       Fault   |                                    |
| |  I:002      O:051       I:051   |                                    |
| +---] [--------]/[--------]/[----+                                    |
|       02         12         05                                       |
Rung 2:4
|    Jog 0                Axis 0                                Jog Axes |
|    Negative   Jog +     Fault                                 Negative |
|    I:002      O:051     I:051                                 O:051    |
+-+---] [--------]/[--------]/[----+------------------( )-----+
| |     01         11         04   |                              12    |
| |  Jog 1                  Axis 1  |                                    |
| |  Negative   Jog +       Fault   |                                    |
| |  I:002      O:051       I:051   |                                    |
| +---] [--------]/[--------]/[----+                                    |
|       03         11         05                                       |
```

ATTENTION: Do not use the PLC ladder logic shown here for jogging axes in the same PLC program with the ladder logic shown previously for homing axes.

Activating the positive or negative jog input for an axis selects that axis in the motion controller as long as the other axis is not selected. To ensure that this axis (and only this axis) remains selected through the release of the jog input, the select bit is latched through the locked bit for the axis (the lock bit is cleared as soon as the jog starts and is set only after the axis has stopped).

Activating the positive jog input for either axis also sets the jog selected axes positive bit in the motion controller to start the jog, as long as there is not a fault on the axis and another axis is not jogging in the negative direction. Similar logic is used for the jog selected axes negative bit.

To control jogging of additional axes, duplicate an axis select rung for each additional axis and change the references as appropriate. Add normally-closed contacts from all other axis selects in each axis select rung to ensure that one (and only one) axis can be selected at a time. Add additional parallel paths in the jog axes positive, and the jog axes negative rungs for the other axes.

The PLC ladder logic shown here for jogging axes should not be used in the same PLC program with the simple ladder logic shown previously for homing axes, or improper operation can result. To provide jogging and homing together via the PLC, the jog and home functions must be interlocked with each other. This ensures that homing one axis does not inadvertently jog (or stop jogging) another axis, or vice-versa, because both functions use the same axis select bits.

Stopping Motion with Kill Control

Setting PLC or SLC bit 7 stops all motion on all axes, disables feedback on all axes, sets all servo outputs to zero, de-activates all drive enable outputs, aborts the program, and disables the CPU Watchdog. Use this bit only in an emergency, when the motion controller must be disabled quickly. The stopping rate of the axes depends on the type of amplifier and the specific connections used, because the motion controller is no longer in control of the axes after executing this block.

ATTENTION: Do not use the Kill Control bit in place of a hard-wired emergency stop circuit for your system.

To re-enable the CPU Watchdog after it has been disabled by the Kill Control bit, cycle power to the motion controller or press the front panel RESET button.

Running and Stopping the Program

Setting SLC or PLC bit 4 runs the application program in the motion controller (generated from a GML Commander diagram and previously downloaded). If the application program is already running, the motion controller ignores this input.

Setting SLC or PLC bit 5 stops the application program. If the application program is not running, the motion controller ignores this input.

While the program is running, the application program running dedicated discrete RIO output from the motion controller (SLC input 8; PLC input 10) is set. When the program is not running, this bit is cleared. The PLC should use this bit to verify that the program has started or stopped before clearing the run or stop application program bits. If a runtime fault occurs while the program is running, the program runtime fault discrete RIO output from the motion controller (SLC input 9; PLC input 11) is set and the application program running bit is cleared. See *Dedicated Discrete Outputs* later in this section for more information on these bits.

ATTENTION: If the motion controller powers up with the Run Application Program bit set, the program will start immediately.

These inputs are level-sensitive, not edge-sensitive. Whenever the motion controller sees the run application program bit set—even immediately after power up—and the program is not currently running, it starts running the program. Specifically, if the motion controller powers up with the run application program bit set, the application program is executed even if the motion controller is configured not to automatically run the application program on power up.

Similarly, whenever the motion controller sees the stop application program bit set—even immediately after power up—the program is stopped. Specifically, if the motion controller powers up with the stop application program bit set, the application program is not executed, even if the motion controller is configured to automatically run the application program on power up in this chapter in the General page of the Configure Control Options dialog box.

ATTENTION: Do not set both the Run Application Program bit and the Stop Application Program bit at the same time otherwise unpredictable operation of the motion controller can occur.

The recommended ladder logic for starting and stopping the motion controller application program using a single input to the PLC is shown below. The rack address in the motion controller is set to 5 and the starting group parameter to 0. Thus the dedicated discrete RIO bits appear in I/O group 1 of rack 5 (I/O:051).

```
Program Listing Report            PLC-5/30

Rung 2:0
|            Program                                              Run     |
| Run/Stop   Running                                              Program |
|   I:002    I:051                                                  O:051 |
+----] [------]/[-----------------------------------------------( )--+
|      17       10                                                  04 |
Rung 2:1
|                                                                       |
|             Program                                             Stop    |
|  Run/Stop   Running                                             Program |
|   I:002     I:051                                                 O:051 |
+----]/[-------] [----------------------------------------------( )--+
|      17        10                                                 05 |
```

When the run/stop input to the PLC (`I:002/17`) is activated, the run application program bit in the motion controller is set, starting the program. Then, when the motion controller confirms that the application program is running by setting the application program running bit, the run application program bit is cleared. When the run/stop input is deactivated, the stop application program bit in the motion controller is set, stopping the program. Then, when the motion controller confirms that the application program is no longer running by clearing the application program running bit, the stop application program bit is cleared.

Pausing and Resuming the Program

Setting SLC or PLC bit 6 pauses the application program running in the motion controller (generated from a GML Commander diagram and previously downloaded).

Note: Only the program pauses. Any ongoing machine motion—caused, for example, by a Jog or a Move function block—continues until the end of that program block.

Clearing this bit resumes the application program from where it was paused. If the application program is not running, the motion controller ignores this input.

While the program is paused, the application program running dedicated discrete RIO output from the motion controller (SLC input 8; PLC input 10) remains set. See *Dedicated Discrete Outputs* later in this section for more information on the application program running bit.

Using the Dedicated Discrete Outputs

RIO Adapter Dedicated Discrete Outputs
I/O Group: Starting Group + 1

Input		
SLC	**PLC**	**Function**
0	0	Axis 0 Locked
1	1	Axis 1 Locked
2	2	Axis 2 Locked
3	3	Axis 3 Locked
4	4	Axis 0 Fault
5	5	Axis 1 Fault
6	6	Axis 2 Fault
7	7	Axis 3 Fault
8	10	Application Program Running
9	11	Program Runtime Fault
10	12	Block Transfer Read Pending
11	13	Illegal Command in Block Transfer

These outputs allow the PLC to monitor the operation of the application program (generated from the GML Commander diagram and downloaded to the motion controller) and the basic status of the axes in the motion controller.

Axis Locked

The axis locked output for each axis (SLC or PLC inputs 0 – 3) is set whenever both the axis is locked onto its command position and no faults are active on the axis. Otherwise, axis locked output is cleared.

An axis is locked whenever either of the following two sets of conditions are true:

- Electronic gearing is OFF, and no position-lock cam or interpolated motion is in progress:
  - No motion (move, jog, or time-lock cam) in progress.
  - Position error ≤ position lock tolerance.
- Electronic gearing is ON, or position-lock cam or interpolated motion is in progress:
  - Position error ≤ position lock tolerance.
  - Ramp to Master Speed is selected in the Electronic Gearing block.

The position lock tolerance is set in the Positioning page of the Configure Axis Use dialog box and specifies how much position error the motion controller tolerates in a locked condition. As such, it is one of the factors that determines positioning accuracy. See the *Installation and Setup Manual* for your controller, or *Online Help* for more information on position lock tolerance.

These bits are identical to the Lock_status Axes System variables in GML Commander. See the *Status Variables* chapter of this manual for more information on Lock_status.

Axis Fault

The axis fault output for each axis (SLC or PLC inputs 4 – 7) is set whenever an axis fault occurs (`Axis_fault` > 0). The next table shows the axis faults that trigger an axis fault output, and the corresponding message that appears in the Terminal Window (or Runtime Display, if enabled).

Axis Faults

Description	On Watch & Axis System Variable Display	Runtime Display
Drive Fault	Drive_fault	DRV FLT
Position Error Tolerance Exceeded	Position_error_fault	ERR FLT
Hardware Overtravel Fault	Hardware_overtravel_fault	HRD LIM
Software Travel Limits Exceeded	Software_overtravel_fault	SFT LIM
Encoder Noise Fault	Encoder_noise_fault	ENC FLT
Encoder Loss Fault	Encoder_loss_fault	ENC FLT

If no axis faults have occurred (Axis_fault = 0), the fault output for the axis is cleared and the message “No Fault” appears in the Terminal Window (or Runtime Display, if enabled.)

Specifically, the axis fault output for an axis is set whenever the corresponding Axis_fault variable in GML Commander has a value greater than 0, and cleared whenever it has a value of 0. See the *Status Variables* chapter of this manual for more information on Axis_fault.

To determine which fault is active when one of the axis fault bits is set, the PLC should issue a block transfer read of the appropriate axis fault status variable in the motion controller. See *Using Block Transfers* in this chapter for more information.

Application Program Running

Running the application program in the motion controller (generated from a GML Commander diagram and previously downloaded) sets either SLC input bit 8, or PLC input bit 10. This bit is cleared when the program stops running. The PLC can use this bit to:

- Verify that the program has started or stopped before clearing the run or stop application program bits.
- Disable block transfers when the program is not running.

See *Running and Stopping the Program* in this chapter and *Using Block Transfers* in this chapter for more information.

Specifically, the application program running output is set whenever the Program_status General System variable in GML Commander has a value greater than 0, and is cleared whenever it has a value of 0. See the *Status Variables* chapter of this manual for more information on Program_status.

Application Program Runtime Fault

The application program runtime fault output from the motion controller (SLC input 9; PLC input 11) is set whenever a runtime fault occurs during execution of a program by the motion controller.

Specifically, the application program runtime fault output is set whenever the Runtime_fault General System variable in GML Commander has a value greater than 0, and is cleared whenever this variable has a value of 0. See the *Status Variables* chapter of this manual for more information on Runtime_fault.

Illegal Command in Block Transfer

The illegal command in block transfer output from the motion controller (SLC input 11; PLC input 13) is set if the last block transfer write to the motion controller contained an illegal value. See *Using Block Transfers* in this chapter for more information.

Using Block Transfers

Block transfers are used to transfer user variable, data parameter, data bit, and cam table values between a data file in the PLC and the motion controller. A block transfer write (BTW) sends data from the PLC to the motion controller, while a block transfer read (BTR) gets data from the motion controller and stores it in the PLC. Up to 60 consecutive user variable or cam table values can be sent or gotten in each block transfer depending on the specified data format.

As long as the block transfers are not interlocked with the motion controller's program running bit in the PLC ladder, a program need not be running in the motion controller for block transfers to work properly. In fact, it is often easier to verify RIO operation using the GML Commander's online interface—selecting the RIO Adapter tag, or using General Watch—to directly examine the variable values once they have been updated by the block transfer.

Sending Data to the Motion Controller

A block transfer write (BTW) is used to send data from the PLC to the motion controller. For example, the PLC rung below executes a block transfer write that sends the data in PLC data file N13 to the motion controller when a switch (I:002/00) is closed.

ATTENTION: The interlocks shown for the BTW are the minimum required for reliable operation. Other conditions can be required in your program.

```
Program Listing Report            PLC-5/30

Rung 2:0

EXAMPLE OF SENDING DATA TO THE MOTION CONTROLLER

| BLOCK                                          BLOCK               |
| TRANSFER                    S CLASS            TRANSFER            |
| SEND DATA   BTW       BTR   PROGRAM            WRITE               |
| CONDITION   ENABLE    ENABLE RUNNING           CONTROL             |
|   I:002     N10:0     N10:5     I:011    +BTW-----------------+     |
+----] [------]/[-------]/[-------] [------+BLOCK TRNSFR WRITE  +-(EN)-+
|      00       15        15        10     |Rack              01|     |
|                                          |Group              0+-(DN) |
|                                          |Module             0|     |
|                                          |Control Block  N10:0+-(ER) |
|                                          |Data file      N13:0|     |
|                                          |Length             0|     |
|                                          |Continuous         N|     |
|                                          +--------------------+     |
```

Other interlocks in the BTW rung can be required for your application. At a minimum, you should have all of the following:

- A condition that causes the BTW (don't send a BTW on every PLC scan).
- The enable bits (Bit 15 of the BT control block) of all other block transfers in the ladder.

The block transfer length should always be set to zero, as shown in the ladder above. This causes all 64 words of the data file to be sent. In addition, continuous block transfers should not be used.

Using the enable bits from each block transfer to interlock all other block transfers ensures that the block transfers always occur in the order in which they appear in the PLC ladder. This is the same technique used to interlock other RIO devices.

Interlocking the block transfers with the program running bit stops the block transfers when the motion controller's program is not running. This significantly improves the downloading of GML Commander diagrams, cam tables, etc., as well as the operation of the motion controllers built-in setup menus. See *Using the Dedicated Discrete Outputs* in this chapter for more information on the program running bit.

ATTENTION: Be sure to specify the correct BTW file type for the numeric format of the transferred values.

The data, that is to be sent to the motion controller, must be stored in a data file in the PLC—N13 in the example above—before the BTW is executed. See *Constructing the PLC Data File* in this chapter. The type of data file, used to store the values to be sent by the BTW, depends on the Numeric Format specified for the transfer, as shown in the table below.

RIO Adapter Block Transfer Write Data File Types

Numeric Format	PLC File Type	
32-bit Integer	Integer	(N)
16-bit Integer	Integer	(N)
32-bit Signed BCD	BCD	(D)
32-bit Floating-Point	Integer*	(N)
Word-Swapped 32-bit Integer	Integer	(N)
Word-Swapped 32-bit Signed BCD	BCD	(D)
Word-Swapped32-bit Floating-Point	Integer*	(N)

* Floating-point values must be copied from a floating bit file to an integer file in the PLC before being block transferred.

Getting Data from the Motion Controller

A block transfer write (BTW), followed immediately by a block transfer read (BTR), is used to get (read) data from the motion controller. The BTW specifies the data to be read; the BTR reads the current values and stores them in a data file in the PLC. Every BTR must be preceded with a BTW to set up the values to be read.

For example, the PLC rungs below execute a BTW to request the data specified in PLC data file N14 from the motion controller, and a BTR to read the data from the motion controller and store it in PLC data file N7 when a switch (I:002/01) is closed. The specified data is read continuously while the switch is closed.

```
Program Listing Report          PLC-5/30

Rung 2:0
EXAMPLE OF GETTING DATA FROM THE MOTION CONTROLLER.
| BLOCK                                                 BLOCK             |
| TRANSFER                          S CLASS             TRANSFER          |
| GET DATA    BTW       BTR       PROGRAM             WRITE             |
| CONDITION   ENABLE    ENABLE    RUNNING             CONTROL           |
|   I:002     N10:0     N10:5     I:011    +BTW-----------------+       |
+----] [------]/[-------]/[-------] [------+BLOCK TRNSFR WRITE  +-(EN)-+
|      01       15        15        10     |Rack               01|      |
|                                          |Group               0+-(DN) |
|                                          |Module              0|      |
|                                          |Control Block  N10:0+-(ER) |
|                                          |Data file      N14:0|      |
|                                          |Length              0|      |
|                                          |Continuous          N|      |
|                                          +--------------------+      |
Rung 2:1
ONCE THE BTW HAS REQUESTED THE DATA, A BTR IS PERFORMED TO READ IT.
| BLOCK                                                                 |
| TRANSFER                          S CLASS                             |
| GET DATA    BTW       BTR       PROGRAM                             |
| CONDITION   ENABLE    ENABLE    RUNNING                             |
|   I:002     N10:0     N10:5     I:011    +BTR-----------------+       |
+----] [------]/[-------]/[-------] [------+BLOCK TRNSFR READ   +-(EN)-+
|      01       15        15        10     |Rack               01|      |
|                                          |Group               0+-(DN) |
|                                          |Module              0|      |
|                                          |Control Block  N10:5+-(ER) |
|                                          |Data file       N7:0|      |
|                                          |Length              0|      |
|                                          |Continuous          N|      |
|                                          +--------------------+      |
```

ATTENTION: The interlocks shown for the block transfers are the minimum required for reliable operation. Other conditions can be required in your application.

Other interlocks in the block transfer rungs can be required for your application. At a minimum, you should have the following:

- A condition that causes the block transfers (don't send block transfers on every PLC scan).
- The enable bits (Bit 15 of the BT control block) of all other block transfers in the ladder.
- The program running bit from the motion controller (SLC input 8; PLC input 10), as shown previously.

The length of the BTW should always be set to zero, as shown. This causes all 64 words of the data file to be sent to the motion controller. In addition, continuous block transfers should not be used.

After the BTW requesting the data is sent, the next block transfer must be a BTR to read the data. If not, the motion controller ignores the second BTW, causing an error in the PLC. To ensure that the PLC and the motion controller do not get out of sync with this write-read-write-read sequence, interlock each block transfer with the enable bits (Bit 15 of the BT control block) of all other block transfers in the ladder, as shown previously. This ensures that the block transfers always occur in the order in that they appear in the PLC ladder.

Interlocking the block transfers with the program running bit stops the block transfers when the motion controller's program is not running. This significantly improves the downloading of GML Commander diagrams, cam tables, etc., as well as the operation of the motion controllers built-in setup menus. See *Using the Dedicated Discrete Outputs* in this chapter for more information on the program running bit.

The data to be read from the motion controller is specified in a data file in the PLC (N14 in the previous example), as explained in *Constructing the PLC Data File* in this chapter. The data returned by the BTR must be stored in another data file (N7 in the example above) in the PLC. After completing the BTR, this data file contains the returned values.

ATTENTION: Be sure to specify the correct BTR file type for the numeric format of the transferred values.

The type of data file used to store the values returned by the BTR depends on the numeric format specified for the transfer, as shown in the table below.

RIO Adapter Block Transfer Write Data File Types

Numeric Format	PLC File Type	
32-bit Integer	Integer	(N)
16-bit Integer	Integer	(N)
32-bit Signed BCD	BCD	(D)
32-bit Floating-Point	Integer*	(N)
Word-Swapped 32-bit Integer	Integer	(N)
Word-Swapped 32-bit Signed BCD	BCD	(D)
Word-Swapped 32-bit Floating-Point	Integer*	(N)

* Floating-point values must be block transferred into an integer file and then copied to a floating bit file in the PLC.

The format of the returned data file also depends on the numeric format specified for the transfer, as explained in *Number and Format of Items* in this chapter.

Constructing the PLC Data File

A BTW data file must be constructed in the PLC, regardless whether the PLC is sending data to or getting data from the motion controller. This data file specifies the type of data (variable, data parameter, etc.), the specific data item (which variable, which data parameter, etc.), or the first item if transferring multiple user variables or cam table values, the number of items, and the format of the values to be read or written. The first three words of the data file are used for this information.

If the PLC is sending data to the motion controller, the remainder of the data file contains the values of the specified items, and the last word is the end-of-block delimiter (000D hex). If the PLC is getting data from the motion controller, no data values need be specified and so the end-of-block delimiter is word 3. Note that the end-of-block delimiter is always the last word used in the data file, which is not necessarily the last possible word in the data file.

Up to 60 sequential user variables or cam table points can be transferred in a single block transfer, depending on the numeric format specified for the transfer. In addition, multiple non-sequential user variables, system variables, data parameters, or data bit values can be transferred in a single block transfer using the intermediate delimiter (003A hex) to concatenate multiple individual transfers into a single BTW. Construct the BTW data file in the PLC as shown below:

RIO Adapter Block Transfer Write Data File Format

Word	Description	
0	Data Type:	7B*at* (hex) to send data; 7D*at* (hex) to read data
	where:	*a* is the Axis (0 - 4, 8, or 9), *t* is the Data Type: 0 = User Variable 1 = Data Parameter 2 = Data Bit 3 = Master Cam Table Position Point 4 = Master Cam Table Time Point 5 = Slave Cam Table Position Point 6 = System Variable (read data only)

RIO Adapter Block Transfer Write Data File Format

Word	Description
1	First Data Item: *t*(Word0)=0: User Variable Number *t*(Word0)=1: Data Parameter Number *t*(Word0)=2: Data Bit Number *t*(Word0)=3: Master Cam Table Point Number *t*(Word0)=4: Master Cam Table Point Number *t*(Word0)=5: Slave Cam Table Point Number *t*(Word0)=6: Internal Variable (all hex values): 0000 = Fault Code 0001 = Analog Input Voltage 0002 = Last Keypress 0003 = Actual Position 0004 = Command Position 0005 = Position Error 0006 = distance To Go 0007 = Marker Distance 0008 = Registration Position 0009 = Strobed Position 000A = Watch Position 000B = Current Task 000C = Status Code 000D = Free Running Clock 000E = Command Velocity 000F = Average Velocity 0010 = Servo Output Level 0011 = Axis 0 Soft Registration Position 0012 = Axis 1 Soft Registration Position 0013 = Axis 2 Soft Registration Position 0014 = Axis 3 Soft Registration Position 0015 = Imaginary Axis Soft Registration Position 0016 = Virtual Axis 0 Soft Registration Position 0017 = Virtual Axis 0 Soft Registration Position 0018 = PCAM Registration Error 0019 = PCAM Average Registration Error 001A = PCAM Registration Error 001B = PCAM Missing Registration Count 001C = PCAM Bad Registration Count 001D = AxisLink Nodes Active 001E = CPU Utilization

RIO Adapter Block Transfer Write Data File Format

Word	Description
2	Number and Format of Items: *fdnn* where *f* is the Numeric Format: 0 = 32-bit integer 1 = 16-bit integer 2 = 32-bit signed BCD 3 = 32-bit floating point 4 = Word-swapped 32-bit integer 5 = Reserved (do not use) 6 = Word-swapped 32-bit signed BCD 7 = Word-swapped 32-bit floating point *d* is the Number of Decimal Digits (*d* 2 OF hex) *nn* is the Number of Sequential Items (*nn*<60 dec)
3-X	Data Item Values (Send data only)
2N+3	End-Of-Block Delimiter (000D hex)

X+1 End-of-Block Delimiter (000D hex) Data Types

The available data types (shown in the following table) are both for sending data to, and getting data from the motion controller. Insert the appropriate value from the table in word 0 of the BTW data file.

ATTENTION: Do not use values other than those shown here in Word 0 of the BTW Data File.

RIOI Adapter Block Transfer Write Data Types (Data File Word 0)

Description	Send Data (Hex)	Get Data (Hex)
User Variable	7B00	7D00
Axis 0 Data Parameter	7B01	7D01
Axis 0 Data Bit	7B02	7D02
Axis 0 Master Cam Position Point*	7B03	7D03
Axis 0 Master Cam Time Point*	7B04	7D04
Axis 0 Slave Cam Position Point*	7B05	7D05
Axis 0 System Variable	—	7D06
Axis 1 Data Parameter	7B11	7D11
Axis 1 Data Bit	7B12	7D12
Axis 1 Master Cam Position Point*	7B13	7D13
Axis 1 Master Cam Time Point*	7B14	7D14
Axis 1 Slave Cam Position Point*	7B15	7D15
Axis 1 System Variable	—	7D16
Axis 2 Data Parameter	7B21	7D21
Axis 2 Data Bit	7B22	7D22
Axis 2 Master Cam Position Point*	7B23	7D23
Axis 2 Master Cam Time Point*	7B24	7D24
Axis 2 Slave Cam Position Point*	7B25	7D25
Axis 2 System Variable	—	7D26
Axis 3 Data Parameter	7B31	7D31
Axis 3 Data Bit	7B32	7D32
Axis 3 Master Cam Position Point*	7B33	7D33
Axis 3 Master Cam Time Point*	7B34	7D34

RIOI Adapter Block Transfer Write Data Types (Data File Word 0)

Description	Send Data (Hex)	Get Data (Hex)
Axis 3 Slave Cam Position Point*	7B35	7D35
Axis 3 System Variable	—	7D36
Imaginary Axis (4) Data Parameter	7B41	7D41
Imaginary Axis (4) Data Bit	7B42	7D42
Imaginary Axis Master Cam Position Point*	7B43	7D43
Imaginary Axis Master. Cam Time Point*	7B44	7D44
Imaginary Axis Slave Cam Position Point*	7B45	7D45
Imaginary Axis System Variable	—	7D46
Virtual Axis 0 Data Parameter	7B81	7D81
Virtual Axis 0 Data Bit	7B82	7D82
Virtual Axis 0 Master Cam Position Point*	7B83	7D83
Virtual Axis 0 System Variable	—	7D86
Virtual Axis 1 Data Parameter	7B91	7D91
Virtual Axis 1 Data Bit	7B92	7D92
Virtual Axis 1 Master Cam Pos. Point*	7B93	7D93
Virtual Axis 1 System Variable	—	7D96

* Indicates that sequential values can be accessed in one block transfer.

For example, to send a position value for axis 2 from the PLC to the master cam table in the motion controller, put 7B23 in word 0 of the BTW data file.

First Data Item

When transferring only one item, the specific item (of the type specified by word 0 of the data file) is specified in word 1 of the data file. When transferring multiple user variables or cam table points in a single block transfer, the first (lowest numbered) item is specified in word 1, as shown in the table below. Put the appropriate value from the table in word 1 of the BTW data file.

RIO Adapter Block Transfer Write First Data Item (Data File Word 1)

Word 0 (Hex)	Word 1 Description	Word 1 Value Range	
		Hex	Decimal
7x00	User Variable Number (V3.0 or later firmware)	0 – 03E7 0 – 07CF	0 – 999 0 – 1,999
7xx1	Data Parameter Number	0 – 0081	0 – 129
7xx2	Data Bit Number	0 – 003C	0 – 60
7xx3	Master Cam Position (V3.0 or later firmware)	0 – 07CF 0 – 32C7	0 – 1,999 0 – 12,999
7xx4	Master Cam Table (V3.0 or later firmware)	0 – 07CF 0 – 32C7	0 – 1,999 0 – 12,999
7xx5	Slave Cam Table Point (V3.0 or later firmware)	0 – 07CF 0 – 32C7	0 – 1,999 0 – 12,999
7Dx6	System Variable	0 – 0010	0 – 16

When more than one user variable or cam table value is sent to or gotten from the motion controller in a single block transfer, word 1 of the BTW data file specifies the first item to be transferred. See *Number and Format of Items* for more information on specifying the number of items.

For example, to send a position value for axis 2 from the PLC to point 100 in the master cam table in the motion controller:

1. Put 7B23 in word 0 of the BTW data file.

2. Put 0064 (100 decimal = 64 hex) in word 1 of the BTW data file.

See detailed information in this chapter on sending or getting each type of item.

Number and Format of Items

Each digit in the hexadecimal representation of word 2 in the BTW data file is used to represent a different part of the format of the values sent to or read from the motion controller. The numeric format of the values is specified by the most significant hex digit of word 2 in the BTW data file. The number of decimal digits (digits to the right of the decimal point) is specified by the next most significant digit. The number of values to send to or get from the motion controller is specified by the two least significant digits of word 2. Thus, word 2 can be represented as *fdnn*, where:

f = Numeric Format of Items
d = Number of Decimal Digits
nn = Number of Items to Transfer

The table below shows the range of values for each of these parameters.

RIO Adapter Block Transfer Write Number and Format Items (Word 2)

Word 2 Digit	Description	Value Range Hex	Value Range Dec
f	32-bit Integer 16-bit Integer 32-bit Signed BCD 32-bit Floating-Point Word-Swapped 32-bit Integer Word-Swapped 32-bit Signed BCD Word-Swapped 32-bit Floating-Point	0 1 2 3 4 6 7	0 1 2 3 4 6 7
d	Number of Decimal Digits	0 – F	0 – 15
nn	Number of Items	1 – 3C	1 – 60

For example, to send 10 position values for axis 2, and each in 32-bit integer format with three digits to the right of the decimal point, and from the PLC to points 100-109 in the master cam table in the motion controller:

1. Put 7B23 in word 0 of the BTW data file.
2. Put 0064 (100 decimal = 64 hex) in word 1 of the data file.
3. Put 030A (32-bit integer; 3 decimal digits; 10 decimal = 0A hex items) in word 2 of the BTW data file.

See detailed information in this chapter on sending or getting each type of item.

Note: Always specify 01 in the low byte of word 2 of the BTW data file, when transferring a Data Parameter, Data Bit, or System Variable.

Up to 60 sequential user variables or cam table points can be transferred in any one block transfer depending on the numeric format specified for the transfer. Sequential data parameter, data bit, or system variables can not be sent to or gotten from the motion controller in one block transfer. To access multiple items of these types, multiple block transfers are required.

The following sections explain each of the available numeric formats.

32-bit Integer Format

When the most significant digit of word 2 in the BTW data file is 0, values are transferred as 32-bit 2s complement integers. In the PLC data file, each 32-bit value is stored as two contiguous 16-bit integers.

If the PLC is sending data to the motion controller, the remainder of the BTW data file (words 3 through 2*nn*+2, where *nn* is the number of items to be sent) contains the values of the specified items, and the last word (word 2*nn*+3) is the end-of-block delimiter, as shown below.

RIO Adapter Block Transfer Write
32-bit Integer Data File Format

Word	Description
0	Data Type
1	First Data Item
2	Number and Format of Items: 0*dnn*
3	First Data Item Value MSW
4	First Data Item Value LSW
5	Second Data Item Value MSW
6	Second Data Item Value LSW
2*nn*+1	Last Data Item Value MSW
2*nn*+2	Last Data Item Value LSW
2*nn*+3	End-of-Block Delimiter (000D hex)

When received by the motion controller, the two integers representing each value are concatenated to form a single 32-bit integer. This integer is then converted to a floating-point value by dividing the received integer values by:

$$10^{\text{(Number of Decimal Digits)}}$$

The values sent to the motion controller must be multiplied by this value before being stored in the BTW data file. Failure to do so results in the transfer of incorrect values to the motion controller.

In 32-bit integer numeric format, the range of values that can be represented is -2,147,483,648 to +2,147,483,647. The number of decimal digits specified for the transfer also limits the largest values that can be sent to the motion controller. The table below shows the factor by which the values in the PLC must be multiplied before being stored in the data file as well as the corresponding maximum representable values in the motion controller.

RIO Adapter Block Transfer Write
32-bit Integer Numeric Format

Number of Decimal Digits:	Multiply PLC Values by:	Maximum Representable Value in the Motion Controller:
0	1	±2,147,483,647.0
1	10	±214,748,364.7
2	100	±21,474,836.47
3	1,000	±2,147,483.647
4	10,000	±214,748.364 7
5	100,000	±21,474.836 47
6	1,000,000	±2,147.483 647
7	10,000,000	±214.748 364 7
8	100,000,000	±21.474 836 47
9	1,000,000,000	±2.147 483 647
10	10,000,000,000	±0.214 748 364 7
11	100,000,000,000	±0.021 474 836 47
12	1,000,000,000,000	±0.002 147 483 647
13	10,000,000,000,000	±0.000 214 748 364 7
14	100,000,000,000,000	±0.000 021 474 836 47
15	1,000,000,000,000,000	±0.000 002 147 483 647

In the previous table, the number of decimal digits is shown as a decimal number. For example, if three decimal digits are specified in the BTW data file (word 2 = 03xx hex) and the value of the item to be sent to the motion controller is 1572, the value 1,572,000 (1572 x 1000) must be stored in the data file. Specifically, since 1,572,000 decimal is 17FCA0 hex, 0017 is stored in the lower numbered word of the word pair (the most significant word), and FCA0 in the higher numbered word (the least significant word).

Up to 30 sequential user variables or cam table points can be transferred in any one block transfer using 32-bit integer numeric format since each block transfer is limited to a maximum of 64 words, as shown by the following formula:

$$30 \text{ Items} \times 2\frac{\text{Words}}{\text{Item}} + 3 \text{ Header Words} + \text{End-of-Block Word}$$

$$= 64 \text{ Words}$$

If the PLC is getting data from the motion controller, no item values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* = 0). The returned data is stored in the integer (N) file specified in the BTR. After completing the BTR, this file contains the returned values, as shown below.

RIO Adapter Block Transfer Read
32-bit Integer Data File Format

Word	Description
0 1	First Data Item Value Most Significant Word First Data Item Value Least Significant Word
2 3	Second Data Item Value Most Significant Word Second Data Item Value Least Significant Word
… …+1	Next Data Item Value Most Significant Word Next Data Item Value Least Significant Word
2*nn*+1 2*nn*+2	Last Data Item Value Most Significant Word Last Data Item Value Least Significant Word
2*nn*+3	End-of-Block Delimiter (000D Hex)

The returned values are stored in word pairs 0, 1 through 2*nn* – 2, 2*nn* – 1 (where *nn* is the number of items read) in 2s complement format. The most significant word of the returned value is stored in the lower numbered word of the file and the least significant word of the returned value in the higher numbered word. Because each 32-bit value is stored as two 16-bit words, 2*nn* + 1 words in the file are used.

The returned values are converted from the floating-point format used in the motion controllers to 32-bit 2s complement integers before being sent to the PLC. This conversion is performed in the motion controller by multiplying the floating-point values by the following and truncating any remaining fractional part:

$$10^{(\text{Number of Decimal Digits})}$$

The number of decimal digits (digits to the right of the decimal point) for the returned values is fixed for all items in a given BTR, and is specified by word 2 in the BTW data file used to request the data from the motion controller. To recover the original value, the PLC must divide the received integer value by the following:

$$10^{(\text{Number of Decimal Digits})}$$

For example, if two decimal digits are specified and the value of the item is 3.1415926, the value 314 ($3.1415926 \times 10^2 = 314.159$ (truncated to 314)) is stored in the returned value file. In this case, since 314 is less than 65,535 (FFFF hex), 0000 is stored in the lower numbered word of the word pair (the most significant word), and 314 (013A hex) in the higher numbered word (the least significant word). The PLC must then divide this value by 100 (10^2), with a result of 3.14—the original value accurate to two decimal places.

With 32-bit integer numeric format, the range of values that can be represented is -2,147,483,648 to +2,147,483,647. This limits the largest values for items read from the motion controller based on the number of decimal digits specified for the transfer as shown in the following table.

RIO Adapter Block Transfer Read
32-bit Integer Format Maximum Item Values

Number of Decimal Digits		
Dec	**Hex**	**Maximum Item Value**
0	0	±2,147,483,647.0
1	1	±214,748,364.7
2	2	±21,474,836.47
3	3	±2,147,483.647
4	4	±214,748.364 7
5	5	±21,474.836 47
6	6	±2,147.483 647
7	7	±214.748 364 7
8	8	±21.474 836 47
9	9	±2.147 483 647
10	A	±0.214 748 364 7
11	B	±0.021 474 836 47
12	C	±0.002 147 483 647
13	D	±0.000 214 748 364 7
14	E	±0.000 021 474 836 47
15	F	±0.000 002 147 483 647

Be careful to select the number of decimal digits such that the expected maximum values of items in the motion controller can be represented with the required precision in the PLC.

16-bit Integer Format

RIO Adapter Block Transfer Write
16-bit Integer Data File Format

Word	Description
0 1 2	Data Type First Data Item Number and Format of Items: 1*dnn*
3 4 *nn* + 2	First Data Item Value Second Data Item Value Last Data Item
nn + 3	Value End-of-Block Delimiter (000D hex)

When received by the motion controller, each integer is converted to a floating-point value by dividing the received integer values by the following:

$$10^{(\text{Number of Decimal Digits})}$$

This means that the values to be sent to the motion controller must be multiplied by this value before being stored in the BTW data file. Failure to do so results in the transfer of incorrect values to the motion controller.

With 16-bit integer numeric format, the range of values that can be represented is -32,768 to +32,767. The number of decimal digits specified for the transfer also limits the largest values that can be sent to the motion controller. The following table shows the factor by which the values in the PLC must be multiplied before being stored in the data file as well as the corresponding maximum representable values in the motion controller.

RIO Adapter Block Transfer Write
16-bit Integer Numeric Format

Number of Decimal Digits:	Multiply PLC Values by:	Maximum Representable Value in Motion Controller:
0	1	±32,767.0
1	10	±3,276.7
2	100	±327.67
3	1,000	±32.767
4	10,000	±3.2767
5	100,000	±0.327 67
6	1,000,000	±0.032 767
7	10,000,000	±0.003 276 7
8	100,000,000	±0.000 327 67
9	1,000,000,000	±0.000 032 767
10	10,000,000,000	±0.000 003 276 7
11	100,000,000,000	±0.000 000 327 67
12	1,000,000,000,000	±0.000 000 032 767
13	10,000,000,000,000	±0.000 000 003 276 7
14	100,000,000,000,000	±0.000 000 000 327 67
15	1,000,000,000,000,000	±0.000 000 000 032 767

In the table above, the number of decimal digits is shown as a decimal number. For example, if three decimal digits are specified in the BTW data file (word 2 = 13xx hex) and the value of the item to be sent to the motion controller is 15.72, the value 3D68 hex ($15.72 \times 10^3 = 15{,}720$ decimal = 3D68 hex) must be stored in the data file.

Up to 60 sequential user variables or cam table points can be transferred in any one block transfer, using 16-bit integer numeric format since each block transfer is limited to a maximum of 64 words, as shown by the formula below:

$$60 \text{ Items} \times 1\frac{\text{Word}}{\text{Item}} + 3 \text{ Header Words} + \text{End-of-Block Word}$$

$$= 64 \text{ Words}$$

If the PLC is getting data from the motion controller, no item values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* = 0). The returned data is stored in the integer (N) data file specified in the BTR. After completing the BTR, this file contains the returned values, as shown below.

RIO Adapter Block Transfer Read
16-bit Integer Returned Values Data File Format

Word	Description
0 1 2	First Data Item Value Second Data Item Value Third Data Item Value
nn-1	Last Data Item Value
nn	End-of-Block Delimiter (000D Hex)

The returned values are stored in words 0 through *nn* – 1 (where *nn* is the number of items read) in 2s complement format. Because each value is stored as a single 16-bit word, *nn* words in the file are used.

The returned values are converted from the floating-point format used in the motion controllers to 16-bit 2s complement integers before being sent to the PLC. This conversion is performed in the motion controller by multiplying the floating-point values by the following and truncating any remaining fractional part:

$$10^{(\text{Number of Decimal Digits})}$$

The number of decimal digits (digits to the right of the decimal point) for the returned values is fixed for all items in a given BTR, and is specified by word 2 in the BTW data file used to request the data from the motion controller. To recover the original value, the PLC must divide the received integer value by:

$$10^{(\text{Number of Decimal Digits})}$$

For example, if two decimal digits are specified and the value of the item is 3.1415926, the value 314 (3.1415926 x 10^2 = 314.159 (truncated to 314 = 013A hex)) is stored in the returned value file. The PLC must then divide this value by 100 (10^2), with a result of 3.14—the original value accurate to two decimal places.

With 16-bit integer numeric format, the range of values that can be represented is -32,768 to +32,767. This limits the largest values for items read from the motion controller based on the number of decimal digits specified for the transfer as shown in the table below.

RIO Adapter Block Transfer Write
16-bit Integer Numeric Format

Number of Decimal Digits		
Dec	**Hex**	**Maximum Item Value**
0	0	±32,767.0
1	1	±3,276.7
2	2	±327.67
3	3	±32.767
4	4	±3.2767
5	5	±0.327 67
6	6	±0.032 767
7	7	±0.003 276 7
8	8	±0.000 327 67
9	9	±0.000 032 767
10	A	±0.000 003 276 7
11	B	±0.000 000 327 67
12	C	±0.000 000 032 767
13	D	±0.000 000 003 276 7
14	E	±0.000 000 000 327 67
15	F	±0.000 000 000 032 767

Be careful to choose the number of decimal digits so the expected maximum values of items in the motion controller can be represented with the required precision in the PLC.

32-bit Signed BCD Format

RIO Adapter Block Transfer Write
32-bit Signed Data File Format

Word	Description
0	Data Type
1	First Data Item
2	Number and Format of Items: 2*dnn*
3	First Data Item Value MSW
4	First Data Item Value LSW
5	Second Data Item Value MSW
6	Second Data Item Value LSW
2*nn*+1	Last Data Item Value MSW
2*nn*+2	Last Data Item Value LSW
2*nn*+3	End-of-Block Delimiter (000D hex)

When received by the motion controller, the two words representing each value are concatenated into the original 8-digit value. This number is then converted to a floating-point value by dividing by:

$$10^{(\text{Number of Decimal Digits})}$$

This means that the values to be sent to the motion controller must be multiplied by this value before being stored in the BTW data file. Failure to do so results in the transfer of incorrect values to the motion controller.

With 32-bit Signed BCD numeric format, the largest value that can be represented is ±79,999,999. The number of decimal digits specified for the transfer also limits the largest values that can be sent to the motion controller. The following table shows the factor by which the values in the PLC must be multiplied before being stored in the data file as well as the corresponding maximum representable values in the motion controller.

RIO Adapter Block Transfer Write
32bit Signed BCD Numeric Format

Number of Decimal Digits:	Multiply PLC Values by:	Maximum Representable Value in Motion Controller:
0	1	±79,999,999.0
1	10	±7,999,999.9
2	100	±799,999.99
3	1,000	±79,999.999
4	10,000	±7,999.999 9
5	100,000	±799.999 99
6	1,000,000	±79.999 999
7	10,000,000	±7.999 999 9
8	100,000,000	±0.799 999 99
9	1,000,000,000	±0.079 999 999
10	10,000,000,000	±0.007 999 999 9
11	100,000,000,000	±0.000 799 999 99
12	1,000,000,000,000	±0.000 079 999 999
13	10,000,000,000,000	±0.000 007 999 999 9
14	100,000,000,000,000	±0.000 000 799 999 99
15	1,000,000,000,000,000	±0.000 000 079 999 999

In the table above, the number of decimal digits is shown as a decimal number. For example, if three decimal digits are specified in the BTW data file (word 2 = 23xx hex) and the value of the item to be sent to the motion controller is 54,321.1234, the value 54,321,123 (54,321.1234 x 10^3 = 54,321,123.4 (truncated to 54,321,123)) must be stored in the data file. Specifically, 5432 is stored in the lower numbered word of the word pair (the most significant word), and 1123 in the higher numbered word (the least significant word).

If the value is negative, the most significant bit of the most significant word must be set to 1. Continuing the example above, the value –54,321.1234 would be stored as D432 hex (5 BCD = 0101 binary; 1101 binary = D hex) in the lower numbered word of the word pair (the most significant word), and 1123 in the higher numbered word (the least significant word) in the BTW data file.

Up to 30 sequential user variables, or cam table points, can be transferred in any one block transfer using 32-bit Signed BCD numeric format, because each block transfer is limited to a maximum of 64 words, as shown by the following formula:

$$30 \text{ Items} \times 2\frac{\text{Words}}{\text{Item}} + 3 \text{ Header Words} + \text{End-of-Block Word}$$

$$= 64 \text{ Words}$$

If the PLC is getting data from the motion controller, no data values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* = 0). The data returned by the BTR is stored in the BCD (D) file specified in the BTR. After completing the BTR, this file contains the returned values, as shown below.

RIO Adapter Block Transfer Read
32-bit Signed BCD Format
Returned Values Data File Format

Word	Description
0 1	First Data Item Value Most Significant Word First Data Item Value Least Significant Word
2 3	Second Data Item Value Most Significant Word Second Data Item Value Least Significant Word
2*nn*-2 2*nn*-1	Last Data Item Value Most Significant Word Last Data Item Value Least Significant Word
2*nn*	End-of-Block Delimiter (000D Hex)

The returned values are stored in word pairs 0, 1 through 2*nn* – 2, 2*nn* – 1 (where *nn* is the number of items read) in signed BCD format. The most significant word of the returned value is stored in the lower numbered word of the data file and the least significant word of the returned value in the higher numbered word. Since each 32-bit value is stored as two 16-bit words, 2*nn* + 1 words in the data file are used.

The returned values are converted from the floating-point format used in the motion controllers to 8-digit BCD values before being sent to the PLC. This conversion is performed in the motion controller by multiplying the floating-point values by the following and truncating any remaining fractional part:

$$10^{(\text{Number of Decimal Digits})}$$

The number of decimal digits (digits to the right of the decimal point) for the returned values is fixed for all items in a given BTR, and is specified by word 2 in the BTW data file used to request the data from the motion controller. To recover the original value, the PLC must divide the received BCD value by this same value.

For example, if two decimal digits are specified and the value of the item is 3.1415926, the value 314 ($3.1415926 \times 10^2 = 314.159$ (truncated to 314)) is stored in the returned value data file. In this case, since 314 is less than 9999, 0000 is stored in the lower numbered word of the word pair (the most significant word), and 0314 in the higher numbered word (the least significant word). The PLC must then divide this value by 100 (10^2), with a result of 3.14—the original value accurate to two decimal places.

If the value is negative, the most significant bit of the most significant word is set to 1. Continuing the example above, the value –3.1415926 would be stored as 8000 (8 = 1000 binary) in the lower numbered word of the word pair (the most significant word), and 0314 in the higher numbered word (the least significant word) in the returned data file.

With 32-bit Signed BCD numeric format, the largest value that can be represented is ±79,999,999. This limits the largest values for items read from the motion controller based on the number of decimal digits specified for the transfer as shown in the following table.

RIO Adapter Block Transfer Read
32-bit Signed BCD Format Maximum Item Values

Number of Decimal Digits		
Dec.	**Hex.**	**Maximum Item Value**
0	0	±79,999,999.0
1	1	±7,999,999.9
2	2	±799,999.99
3	3	±79,999.999
4	4	±7,999.999 9
5	5	±799.999 99
6	6	±79.999 999
7	7	±7.999 999 9
8	8	±0.799 999 99
9	9	±0.079 999 999
10	A	±0.007 999 999 9
11	B	±0.000 799 999 99
12	C	±0.000 079 999 999
13	D	±0.000 007 999 999 9
14	E	±0.000 000 799 999 99
15	F	±0.000 000 079 999 999

Be careful to choose the number of decimal digits such that the expected maximum values of items in the motion controller can be represented with the required precision in the PLC.

32-bit Floating Point Format

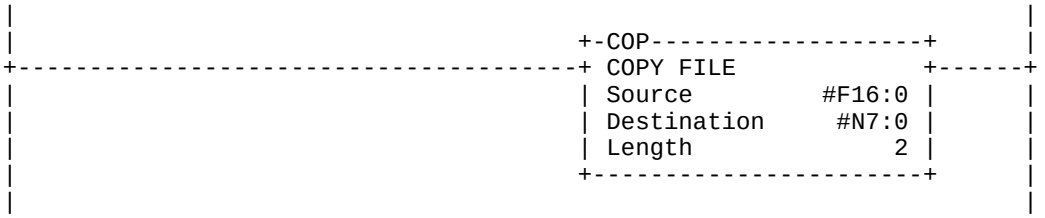
When the most significant digit of word 2 in the BTW data file is 3, values are transferred as IEEE-format 32-bit floating point values. In the PLC data file, each 32-bit floating point value is stored as two contiguous 16-bit integers. File Copy (COP) instructions must be used in the PLC to transfer these two integers from a floating bit (F) file to the integer (N) BTW data file for transfer to the motion controller.

If the PLC is sending data to the motion controller, the remainder of the BTW data file (words 3 through 2*nn*+2, where *nn* is the number of items to be sent) contains the values of the specified items, and the last word (word 2*nn*+3) is the end-of-block delimiter, as shown in the following table.

RIO Adapter Block Transfer Write
32-bit Floating -Point Data File Format

Word	Description
0	Data Type
1	First Data Item
2	Number and Format of Items: 30*nn*
3	First Data Item Value Most Significant Word
4	First Data Item Value Least Significant Word
5	Second Data Item Value Most Significant Word
6	Second Data Item Value Least Significant Word
2*nn*+1	Last Data Item Value Most Significant Word
2*nn*+2	Last Data Item Value Least Significant Word
2*nn*+3	End-of-Block Delimiter (000D hex)

Use a File Copy instruction in the PLC program to transfer the values that you want from a floating bit file to the BTW data file. For example, the following unconditional rung copies word 0 (32 bits) of floating bit file F16 in the PLC to words 0 and 1 (16 bits each) of the integer file N7.



Specifying N7 as the data file for the BTW as shown in *Sending Data to the Motion Controller* in this chapter transfers the 32-bit floating-point value to the motion controller.

When received by the motion controller, the two integers representing each value are concatenated into the original 32-bit floating-point value. Unlike the preceding fixed-point (integer) formats, 32-bit floating-point format does not require specification of the number of decimal digits in the value. Thus, the second most significant digit of Word 2 in the BTW data file (digit *d*) should be set to 0 as shown above. With 32-bit floating-point numeric format, the largest value that can be represented is $\pm 1 \times 10^{38}$ and the smallest is $\pm 1 \times 10^{-38}$.

Up to 30 sequential user variables, or cam table points, can be transferred in any one block transfer using 32-bit floating-point numeric format because each block transfer is limited to a maximum of 64 words, as shown by the formula below:

$$30 \text{ Items} \times 2\frac{\text{Words}}{\text{Item}} + 3 \text{ Header Words} + \text{End-of-Block Word}$$

$$= 64 \text{ Words}$$

If the PLC is getting data from the motion controller, no item values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* = 0). The returned data is stored in the integer (N) file specified in the BTR. After completing the BTR, this file contains the returned values, as shown below.

RIO Adapter Block Transfer Read
32-bit Floating-Point Returned Values Data File Format

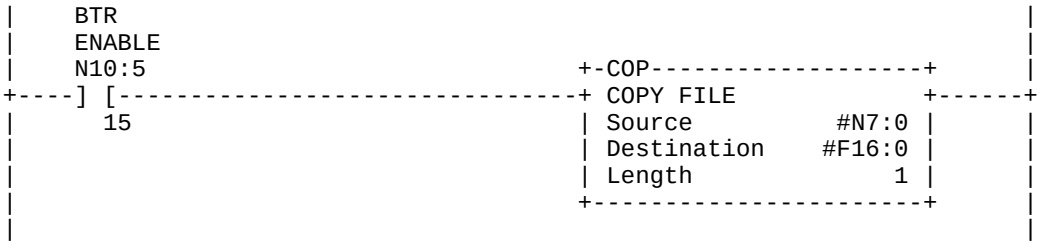
Word	Description
0 1	First Data Item Value Most Significant Word First Data Item Value Least Significant Word
2 3	Second Data Item Value Most Significant Word Second Data Item Value Least Significant Word
… …+1	Next Data Item Value Most Significant Word Next Data Item Value Least Significant Word
2*nn*-2 2*nn*-1	Last Data Item Value Most Significant Word Last Data Item Value Least Significant Word

RIO Adapter Block Transfer Read
32-bit Floating-Point Returned Values Data File Format

Word	Description
2*nn*	End-of-Block Delimiter (000D Hex)

The returned values are stored in word pairs 0, 1 through 2*nn* – 2, 2*nn* – 1 (where *nn* is the number of items read) as 16-bit integers. The most significant word of the returned value is stored in the lower numbered word of the data file, and the least significant word of the returned value is stored in the higher numbered word. Since each 32-bit value is stored as two 16-bit words, 2*nn* + 1 words in the file are used.

Use a File Copy instruction in the PLC program after the BTR to transfer the values from the integer BTR data file to a floating bit file for use by the PLC. For example, the following rung copies words 0 and 1 (16 bits each) of the integer file N7 (block transferred to the PLC as shown *in Getting Data from the Motion Controller* in this chapter) to word 0 (32 bits) of floating bit file F16.



With 32-bit floating-point numeric format, the largest value that can be represented is ±1 x 10^{38} and the smallest is ±1 x 10^{-38}.

Word-Swapped 32-bit Integer Format

When the most significant digit of word 2 in the BTW data file is 4, values are transferred as 32-bit 2s complement integers. In the PLC data file, each 32-bit value is stored as two contiguous 16-bit integers. This format is identical to the 32-bit integer format described earlier except the order of the words in the data files. See *32-bit Integer Format* in this chapter for information on the range, precision, and conversion of values.

If the PLC is sending data to the motion controller, the remainder of the BTW data file (words 3 through 2*nn*+2, where *nn* is the number of items to be sent) contains the values of the specified items, and the last word (word 2*nn*+3) is the end-of-block delimiter, as shown below.

RIO Adapter Block Transfer Write
Word-Swapped 32-bit Integer Data File Format

Word	Description
0	Data Type
1	First Data Item
2	Number and Format of Items: 4*dnn*
3	First Data Item Value Least Significant Word
4	First Data Item Value Most Significant Word
5	Second Data Item Value Least Significant Word
6	Second Data Item Value Most Significant Word
2*nn*+1	Last Data Item Value Least Significant Word
2*nn*+2	Last Data Item Value Most Significant Word
2*nn*+3	End-of-Block Delimiter (000D hex)

If the PLC is getting data from the motion controller, no item values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* = 0). The returned data is stored in the integer (N) file specified in the BTR. After completing the BTR, this file contains the returned values, as shown below.

RIO Adapter Block Transfer Read
Word-Swapped 32-bit Integer Returned Values Data File Format

Word	Description
0	First Data Item Value Least Significant Word
1	First Data Item Value Most Significant Word
2	Second Data Item Value Least Significant Word
3	Second Data Item Value Most Significant Word
…	Next Data Item Value Least Significant Word
…+1	Next Data Item Value Most Significant Word

RIO Adapter Block Transfer Read
Word-Swapped 32-bit Integer Returned Values Data File Format

Word	Description
2*nn*-2 2*nn*-1	Last Data Item Value Least Significant Word Last Data Item Value Most Significant Word
2*nn*	End-of-Block Delimiter (000D Hex)

The returned values are stored in word pairs 0, 1 through 2*nn* – 2, 2*nn* – 1 (where *nn* is the number of items read) in 2s complement format. The least significant word of the returned value is stored in the lower numbered word of the file and the most significant word of the returned value in the higher numbered word. Since each 32-bit value is stored as two 16-bit words, 2*nn* + 1 words in the file are used.

Word-Swapped 32-bit Signed BCD Format

When the most significant digit of word 2 in the BTW data file is 6, values are transferred as signed 32-bit (8-digit) BCD numbers. In the PLC data file, each 32-bit value is stored as two contiguous 16-bit integers. The sign bit is the most significant bit of the most significant word and is 0 if the value is positive and 1 if it is negative. This format is identical to the 32-bit signed BCD format described earlier except for the order of the words in the data files. See *32-bit Signed BCD Format* in this chapter for information on the range, precision, and conversion of values.

If the PLC is sending data to the motion controller, the remainder of the BTW data file (words 3 through 2*nn*+2, where *nn* is the number of items to be sent) contains the values of the specified items, and the last word (word 2*nn*+3) is the end-of-block delimiter, as shown below.

RIO Adapter Block Transfer Write
Word-Swapped 32-bit Signed BCD Data File Format

Word	Description
0	Data Type
1	First Data Item
2	Number and Format of Items: 6*dnn*

RIO Adapter Block Transfer Write
Word-Swapped 32-bit Signed BCD Data File Format

Word	Description
3 4	First Data Item Value Least Significant Word First Data Item Value Most Significant Word
5 6	Second Data Item Value Least Significant Word Second Data Item Value Most Significant Word
2*nn*+1 2*nn*+2	Last Data Item Value Least Significant Word Last Data Item Value Most Significant Word
2*nn*+3	End-of-Block Delimiter (000D hex)

If the PLC is getting data from the motion controller, no data values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* - 0). The data returned by the BTR is stored in the BCD (D) file specified in the BTR. After completing the BTR, this file contains the return values, as shown below.

RIO Adapter Block Transfer Read
Word-Swapped 32-bit Signed BCD
Returned Values Data File Format

Word	Description
0 1	First Data Item Value Least Significant Word First Data Item Value Most Significant Word
2 3	Second Data Item Value Least Significant Word Second Data Item Value Most Significant Word
2*nn*-2 2*nn*-1	Last Data Item Value Least Significant Word Last Data Item Value Most Significant Word
2*nn*	End-of-Block Delimiter (000D Hex)

The returned values are stored in word pairs 0, 1 through 2*nn* – 2, 2*nn* – 1 (where *nn* is the number of items read) in signed BCD format. The least significant word of the returned value is stored in the lower numbered word of the data file and the most significant word in the higher numbered word. Since each 32-bit value is stored as two 16-bit words, 2*nn* + 1 words in the data file are used.

Word-Swapped 32-bit Floating Point Format

When the most significant digit of word 2 in the BTW data file is 7, values are transferred as IEEE-format 32-bit floating point values. In the PLC data file, each 32-bit floating-point value is stored as two contiguous 16-bit integers. This format is identical to the 32-bit floating-point format described earlier except for the order of the words in the data files. See *32-bit Floating Point Format* in this chapter for information on the range and precision of values.

ATTENTION: Do not use Word-Swapped 32-bit Floating-Point format to transfer PLC values to or from the motion controller.

Word-swapped 32-bit floating-point values cannot be directly converted to or from floating point format in the PLC using the copy (COP) instruction. This format should not be used for transferring PLC floating point values to or from the motion controller.

If the PLC is sending data to the motion controller, the remainder of the BTW data file (words 3 through 2*nn*+2, where *nn* is the number of items to be sent) contains the values of the specified items, and the last word (word 2*nn*+3) is the end-of-block delimiter, as shown below.

RIO Adapter Block Transfer Write Word-Swapped 32-bit Floating-Point Data File Format

Word	Description
0 1 2	Data Type First Data Item Number and Format of Items: 70*nn*
3 4	First Data Item Value Least Significant Word First Data Item Value Most Significant Word
5 6	Second Data Item Value Least Significant Word Second Data Item Value Most Significant Word
2*nn*+1 2*nn*+2	Last Data Item Value Least Significant Word Last Data Item Value Most Significant Word
2*nn*+3	End-of-Block Delimiter (000D hex)

If the PLC is getting data from the motion controller, no item values need be specified in the BTW data file and thus the end-of-block delimiter is word 3 (*nn* = 0). The returned data is stored in the integer (N) file specified in the BTR. After completing the BTR, this file contains the returned values, as shown in the following table.

RIO Adapter Block Transfer Read Word-Swapped 32-bit Floating-Point Returned Values Data File Format

Word	Description
0 1	First Data Item Value Least Significant Word First Data Item Value Most Significant Word
2 3	Second Data Item Value Least Significant Word Second Data Item Value Most Significant Word
… …+1	Next Data Item Value Least Significant Word Next Data Item Value Most Significant Word
2*nn*-2 2*nn*-1	Last Data Item Value Least Significant Word Last Data Item Value Most Significant Word
2*nn*	End-of-Block Delimiter (000D Hex)

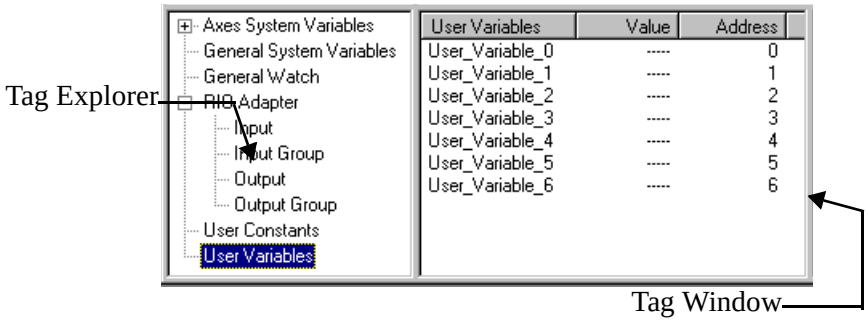
The returned values are stored in word pairs 0, 1 through 2*nn* – 2, 2*nn* – 1 (where *nn* is the number of items read) as 16-bit integers. The least significant word of the returned value is stored in the lower numbered word of the data file and the most significant word of the returned value in the higher numbered word. Because each 32-bit value is stored as two 16-bit words, 2*nn* + 1 words in the file are used.

Sending or Getting User Variable Values

One or more sequentially numbered user variable values can be sent to or gotten from the motion controller using a block transfer. The variable number (or address) of the variable to be transferred (or of the first variable, if sequential values are to be transferred) must be put in word 1 of the BTW data file for the transfer.

To view the appropriate variable address,

1. Open the GML Commander diagram.
2. Select **User Variables** in the Tag Explorer (located in the lower left corner of the GML Commander screen), and the available user variables and their addresses appear in the adjoining Tag Window, as in the following example.



The variable number is the Address listed in the column at the far right.

For example, the BCD data file shown below (in hexadecimal format) sends the values 1, 2, 3, 4, and 5 to user variables 1 through 5 (respectively) in the motion controller, when specified in a BTW. Word 0 specifies that this block transfer is sending (7Bxx) user variables (xx00). Word 1 specifies the first item as user variable 1 (0001). Word 2 specifies 32-bit signed BCD numeric format (2xxx) with two decimal digits (x2xx), and that five variable values are to be sent (xx05). Word pairs 3, 4 through 11, 12 are the values to be sent. Word 13 is the end-of-block delimiter (000D).

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4      5      6      7      8      9
D13:0   7B00   0001  2205   0000   0100   0000   0200   0000   0300   0000
D13:10  0400   0000  0500   000D   0000   0000   0000   0000   0000   0000
D13:20  0000   0000  0000   0000   0000   0000   0000   0000   0000   0000
D13:30  0000   0000  0000   0000   0000   0000   0000   0000   0000   0000
D13:40  0000   0000  0000   0000   0000   0000   0000   0000   0000   0000
D13:50  0000   0000  0000   0000   0000   0000   0000   0000   0000   0000
D13:60  0000   0000  0000   0000
```

The values in the data file have been multiplied by 100 to account for the two decimal digits specified in word 2. For example, to send the value 2 to user variable 2, 0000 is put in word 5 (the most significant word) and 200 BCD in word 6 (the least significant word). See *Constructing the PLC Data File* in this chapter for more information.

Sending or Getting Data Parameter Values

Individual data parameter values for any of the motion controller axes can be sent to or taken from the motion controller using a block transfer. To send or get multiple data parameter values, multiple block transfers (one for each value) must be used. The parameter number of the data parameter to be transferred must be put in word 1 of the BTW data file for the transfer.

- Put the parameter number from the table (just the number, not the D) of the parameter to be transferred in word 1 of the BTW data file for the transfer.
- Put 01 for the number of sequential parameter values to be transferred in the low byte of word 2 of the BTW data file for the transfer.

Refer to the Data Parameters table in *Appendix A* of this manual to determine the appropriate parameter number. In all cases, block transfers affect only the working data parameter values.

Note: An axis must always be specified, even for data parameters that are not axis-specific.

ATTENTION: Do not send out-of-range data parameter values to the motion controller.

Data parameter values sent to the motion controller must always be within the range implied by the value format for the parameter shown in the Data Parameters table for the Control Setting block in *Appendix A* of this manual or unpredictable operation results. In addition, always specify the same number of decimal digits for the transfer as shown to the right of the decimal point in the table.

For example, the integer data file shown below (in hexadecimal format) gets the Maximum Negative Travel value (data parameter 12) for axis 1 from the motion controller when specified in a BTW as shown in *Getting Data from the Motion Controller* in this chapter. Word 0 specifies that this block transfer is getting (7Dxx) data parameters for axis 1 (xx11). Word 1 specifies the first item as data parameter 12 (12 decimal = 000C hex). Word 2 specifies 32-bit floating-point format (30xx) and that one item is to be sent (xx01). Word 3 is the end-of-block delimiter (000D).

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4     5      6      7     8      9
N14:0   7D11   000C  3001   000D   0000  0000   0000   0000  0000   0000
N14:10  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:20  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:30  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:40  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:50  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:60  0000   0000  0000   0000
```

The returned value is stored in data file N7 as two 16-bit integers representing the floating-point value. For example, a Maximum Negative Travel value of –511.4872 is returned as C3FF in word 0 (the most significant word) and BE5D in word 1 (the least significant word) as shown below.

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4     5      6      7     8      9
N7:0    C3FF   BE5D  000D   0000   0000  0000   0000   0000  0000   0000
N7:10   0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N7:20   0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N7:30   0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N7:40   0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N7:50   0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N7:60   0000   0000  0000   0000
```

See *Getting Data from the Motion Controller* in this chapter for more information on the format returned values file.

If words 0 and 1 of the integer data file N7 are copied to word 0 of floating bit file F16 in the PLC using a File Copy instruction (as shown in *32-bit Floating-Point Format* in this chapter) the original floating-point value is restored.

```
Data Table Report      PLC-5/30

Address         0            1            2            3            4
F16:0       -511.4872       0.0          0.0          0.0          0.0
F16:0          0.0          0.0          0.0          0.0          0.0
F16:0          0.0          0.0          0.0          0.0          0.0
F16:0          0.0          0.0          0.0          0.0          0.0
F16:0          0.0          0.0          0.0          0.0          0.0
F16:0          0.0          0.0          0.0          0.0          0.0
F16:0          0.0          0.0
```

Sending or Getting Data Bit Values

Individual data bit values for any of the motion controller axes can be sent to or taken from the motion controller using a block transfer. To send or get multiple data bit values, multiple block transfers (one for each value) must be used. The bit number of the parameter to be transferred must be put in word 1 of the BTW data file for the transfer.

- Put the bit number from the table (just the number, not the B) of the bit to be transferred in word 1.
- Put 01 for the number of bits to be transferred in the low byte of word 2 of the BTW data file for the transfer.

Use the Data Bits table in the *Appendix A* of this manual to determine the appropriate bit number. In all cases, block transfers affect only the working data bit value.

Note: An axis must always be specified (low byte of word 0), even for data bits that are not axis-specific.

Data bit values sent to the motion controller must always be either 1 or 0 or unpredictable operation can result. In addition, since these values are integers, 16-bit integer format with zero decimal digits should always be specified in the high byte of word 2 in the BTW data file.

For example, the integer data file shown below (in hexadecimal format) sets the overtravel switch polarity (data bit 5) for axis 2 to NC (normally closed) when specified in a BTW as shown in *Sending Data to the Motion Controller* in this chapter. Word 0 specifies that this block transfer is sending (7Bxx) data bits for axis 2 (xx22). Word 1 specifies the item as

data bit 5 (0005). Word 2 specifies 16-bit integer format (1xxx), zero decimal digits (x0xx), and that one value is to be sent (xx01). Word 3 is the value to be sent and word 4 is the end-of-block delimiter (000D).

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4     5      6      7     8      9
N13:0   7B22   0005  1001   0001   000D  0000   0000   0000  0000   0000
N13:10  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N13:20  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N13:30  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N13:40  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N13:50  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N13:60  0000   0000  0000   0000
```

The value shown for the data bit (1) specifies normally-closed contacts for the hardware overtravel switches for axis 2. See the Data Bits table in Control Settings in the *Function Blocks* chapter of this manual and your *Installation and Setup* manual for your motion controller for more information on data bits.

Sending or Getting Cam Table Values

One or more sequentially numbered master or slave cam table profile point values can be sent to or taken from the motion controller using a block transfer. Put the appropriate value in word 0 of the BTW data file depending on whether the transfer is sending or getting the following, as explained in *Data Types* in this chapter:

- Position or time values from the master cam table.
- Position values from the slave cam table.

Be sure to specify the same axis in the BTW data file as specified in the Position Lock Cam or Time Lock Cam block that executes the profile, otherwise erroneous values will be transferred.

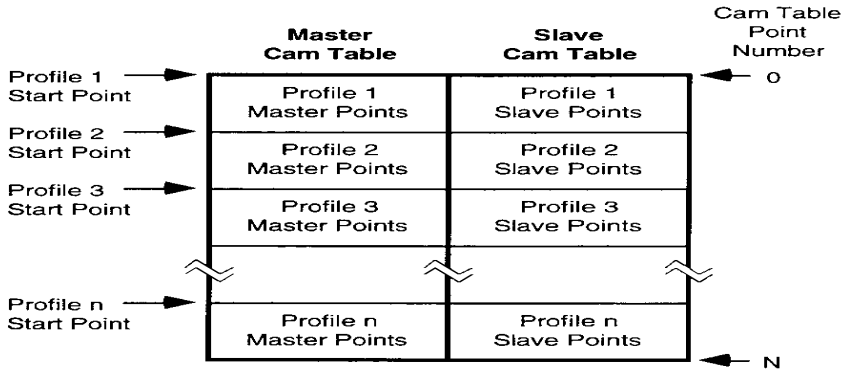
The point number of the point to be transferred (or of the first point, if sequential values are to be transferred) must be put in word 1 of the BTW data file for the transfer.

To view point numbers:

1. Open the GML Commander diagram.
2. Double-click on the Configure Cam block used to execute the appropriate profile. The Configure Cam dialog box opens, displaying the starting and ending points of that profile in the cam tables.

Put the number of sequential cam table values to be transferred in the low byte of word 2 of the BTW data file for the transfer.

Your motion controller contains two cam tables, that can be used to store many individual time-lock and/or position-lock cam profiles. The master cam table stores time values (for time-lock cams) or master axis positions (for position-lock cams), while the slave cam table stores the corresponding slave axis positions, as shown below.



N = 1999 for iCODE V2.3 and earlier;
12,999 for iCODE V 3.0 and later

The cam tables can store any number of individual profiles of any length as long as the total number of points for all profiles does not exceed the capacity of the table. If you selected iCODE version 3.0 or later in the General page of the Configure Control Options dialog box, each cam table can contain up to 13,000 points numbered 0 through 12999. If you selected iCODE version 2.3 or earlier, each cam table can contain up to 2,000 points numbered 0 through 1999.

Note: If you are using the Build Table Block to construct Master and Slave Cam Tables, the Build Table function is limited to a total of 4000 Master and 4000 Slave cam points.

The start point for any individual cam profile is the cam table point number (0 – 1999 or 0 – 12999, depending on iCODE version) of the first point in the appropriate profile. Likewise, the end point is the cam table point number of the last point in that profile.

For example, assuming that profile 2 in the previous figure starts at point 100 in the cam tables, the integer data file shown below (in hexadecimal format) gets the first 30 axis 0 position points (points 100 – 129) from the slave cam table when specified in a BTW as shown in *Getting Data from the Motion Controller* in this chapter. Word 0 specifies that this block transfer is getting (7Dxx) axis 0 slave cam position points (xx05). Word 1 specifies the first item as cam table point 100 (0064 hex). Word 2 specifies 32-bit integer format (0xxx), two decimal digits in the values (x2xx), and that 30 points are to be gotten (xx1E hex). Word 3 is the end-of-block delimiter (000D).

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4     5      6      7     8      9
N14:0   7D05   0064  021E   000D   0000  0000   0000   0000  0000   0000
N14:10  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:20  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:30  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:40  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:50  0000   0000  0000   0000   0000  0000   0000   0000  0000   0000
N14:60  0000   0000  0000   0000
```

The returned values are stored in data file N7 multiplied by 100 to account for the two decimal digits specified in word 2. For example, if slave position point 103 has a value of 436.15, 0000 is returned in word 6 (the most significant word) and AA5F in word 7 (the least significant word) since 436.15 x 100 = 43,615 decimal = 0000 AA5F hex). See *Getting Data from the Motion Controller* in this chapter for more information on the format of returned values.

Getting System Variable Values

Individual system variable values for any of the motion controller axes can be taken from the motion controller using a block transfer. The system variable to be transferred is specified as a value in word 1 of the BTW data file for the transfer as shown in the following table.

RIO Adapter Block Transfer Write System Variable Numbers (Word 1)

System Variable	Variable Number (Hex)	Identical to GML Commander System Variable
Fault Code	0000	—
Analog Input Voltage†	0001	Analog_Input
Last Keypress	0002	Last_Keypress
Actual Position	0003	Actual_Position
Command Position*	0004	Command_Position
Position Error*	0005	Position_Error
Distance To Go*	0006	Distance_To_Go
Marker Distance	0007	Marker_Distance
Registration Position	0008	Registration_Position
Strobed Position	0009	Strobed_Position
Watch Position	000A	Watch_Position
Current Task	000B	Current_Task
Status Code	000C	—
Free Running Clock	000D	Free_Running_Clock
Command Velocity*	000E	Command_Velocity
Average Velocity*	000F	Average_Velocity
Servo Output Level*	0010	Servo_Output_Level
Axis 0 Soft Registration Position	0011	Soft_Reg_Pos_Axis0

RIO Adapter Block Transfer Write System Variable Numbers (Word 1)

System Variable	Variable Number (Hex)	Identical to GML Commander System Variable
Axis 1 Soft Registration Position	0012	Soft_Reg_Pos_Axis1
Axis 3 Soft Registration Position	0013	Soft_Reg_Pos_Axis2
Axis 0 Soft Registration Position	0014	Soft_Reg_Pos_Axis3
Imaginary Axis Soft Reg. Position	0015	Soft_Reg_Pos_Imag
Virtual Axis 0 Soft Reg. Position	0016	Soft_Reg_Pos_Virtual0
Virtual Axis 1 Soft Reg. Position	0017	Soft_Reg_Pos_Virtual1
PCAM Registration Error	0018	PCAM_registration_error
PCAM Average Registration Error	0019	PCAM_average_registration_error
PCAM Good Registration Count	001A	PCAM_good_registration_count
PCAM Missing Registration Count	001B	PCAM_missing_registration_error
PCAM Bad Registration Error	001C	PCAM_bad_registration_error
AxisLink Nodes Active	001D	AxisLink_configuration_nodes
CPU Utilization	001E	CPU_utilization

*Not available for Master Only axes.
† Only available in iCODE firmware V2.3 and earlier.

An axis must always be specified (in word 0), even for system variables that are not axis-specific (like Last_Keypress). For analog input values (only available in IMC-S/20x-R and IMC/S21x-R model motion controllers), the axis specifies the corresponding analog input (specify axis 0 to read analog input 0, axis 1 for analog input 1, etc.). In addition, since only one system variable value can be transferred in each BTR, always put 01 for the number of items in the low byte of word 2 in the BTW data file for the transfer.

Except for the fault and status codes, the system variable values returned by the BTR are identical to those for the corresponding GML Commander variables shown in the previous table. See the *System Variables* chapter for detailed information on the units and format of the returned values for these system variables.

For example, the data file shown below (in hexadecimal format) retrieves the actual position of axis 0 when specified in a BTW as shown in *Getting Data from the Motion Controller* in this chapter. Word-Swapped 32-bit Signed BCD numeric format is used to allow easy transfer of the value to a PanelView for display. Word 0 specifies that this block transfer is getting (7Dxx) an axis 0 system variable (xx06). Word 1 specifies the variable as the registration position (0008 hex). Word 2 specifies word-swapped 32-bit signed BCD numeric format (6xxx), three decimal digits in the value (x3xx), and that one value is to be gotten (xx01 hex). Word 3 is the end-of-block delimiter (000D).

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4      5      6      7     8      9
D14:0   7D06   0008  6301   000D   0000   0000   0000   0000  0000   0000
D14:10  0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D14:20  0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D14:30  0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D14:40  0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D14:50  0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D14:60  0000   0000  0000   0000
```

The returned value is stored in BCD data file D7 multiplied by 1,000 to account for the three decimal digits specified in word 2. If the current registration position of axis 0 is 436.15, 0043 is returned in word 1 and 6150 in word 0.

```
Data Table Report      PLC-5/30

Address  0      1     2      3      4      5      6      7     8      9
D7:0    6150   0043  000D   0000   0000   0000   0000   0000  0000   0000
D7:10   0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D7:20   0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D7:30   0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D7:40   0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D7:50   0000   0000  0000   0000   0000   0000   0000   0000  0000   0000
D7:60   0000   0000  0000   0000
```

See *Getting Data from the Motion Controller* in this chapter for more information on the format of returned values.

Fault and Status Codes

When getting the fault code for an axis, always specify 1001 hex (for 16-bit integer numeric format, 0 decimal digits, and 1 item) in word 2 of the BTW data file for the transfer.

For the status code, specify 1001 hex (for 16-bit integer numeric format, 0 decimal digits, and 1 item) in word 2 of the BTW data file only if the AxisLink option is not being used. To get all 17 bits of the status code, a numeric format (*f*) of something other than 1 or 5 must be entered in the upper nibble of word 2 (*fxxx*) In addition, for the status code as well as the fault code, the returned value is binary coded and is not the same as the Axis_fault or Axis_status system variable in GML Commander.

Unlike the prioritized scheme used in GML Commander for reporting faults and status (in which the value is equal to the highest priority active condition), a linear bit select scheme is used with RIO block transfers. Thus, each bit position represents a specific fault or status condition and multiple conditions can be reported to the PLC in one BTR. In all cases, if the specific fault or status is true, the bit is ON (1), and OFF (0) if not.

The fault indicated by each bit in the fault code returned value is identical to the GML Commander fault variable shown in the following table.

RIO Adapter Block Transfer Read Fault Code Returned Value Bits

Bit	Is Identical to Variable	Runtime Display
8	(Runtime_fault > 0)	-------
7	AxisLink_failed	-------
6	AxisLink_timeout	-------
5	Encoder_loss_fault*	ENC FLT
4	Encoder_noise_fault	ENC FLT
3	Software_overtravel_fault	SFT LIM
2	Hardware_overtravel_fault	HRD LIM
1	Position_error_fault	ERR FLT
0	Drive_fault	DRV FLT

*Only available in 1394 GMC/GMC Turbo and Compact with V3.X firmware.

For example, a returned fault code value of 0012 hex for an axis indicates that encoder noise has been detected and the position error tolerance has been exceeded (12 hex = 0001 0010 binary). See the *Fault Variables* chapter of this manual for more information on these fault conditions.

The status indicated by each bit in the returned status code value is identical to the GML Commander status variable shown in the following table. Bits not shown in the table are reserved for future use and should not be used as their value is undefined.

RIO Adapter Block Transfer Read Status Code Returned Value Bits

Bit			
Hex.	**Oct**	**Is Identical to Variable . . .**	**Runtime Display**
10	20	AxisLink_status	
F	17	PCAM_auto_correction_status	OUT LIM
E	16	PCAM_pending_profile_status	
D	15	PCAM_profile_status	
C	14	Output_limit_status	
B	13	Watch_Pos_status	
A	12	TCAM_status	MOVING
9	11	Registration_status	
8	10	PCAM_status	
7	7	Move_status	MOVING
6	6	Lock_status	LOCKED
5	5	Jog_status	JOGGING
4	4	Homing_status	HOMING
3	3	Gearing_status	
2	2	Feedback_status	SRV OFF
1	1	Decel_status	
0	0	Accel_status	

For example, a returned status code value of 028C hex for an axis indicates that a registration event for the axis has occurred and that the axis is moving, electronic gearing is enabled, and feedback is ON (028C hex = 0000 0010 1000 1100 binary). See the *Status Variables* chapter of this manual for more information on these conditions.

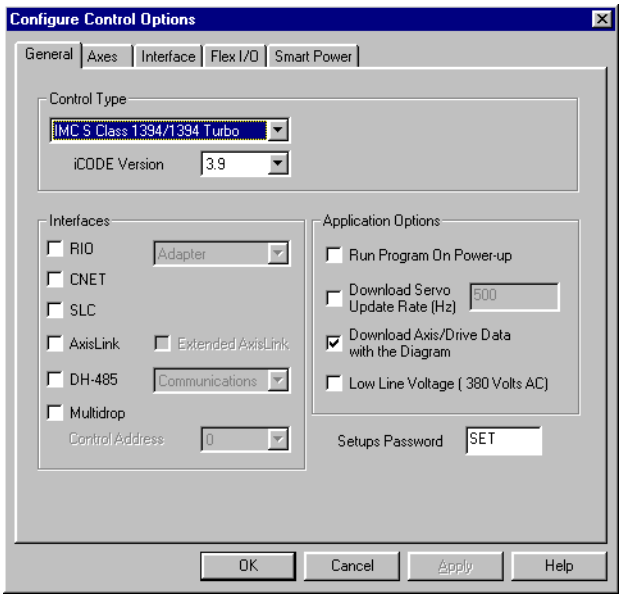
Chapter **35**

Configuring ControlNet (CNET)

Enabling the CNET Interface in GML Commander

To enable the CNET interface in GML Commander:

1. Select **Configure** from the menu bar. The Configure menu displays.
2. Select **Control Options**. The Configure Control Options dialog box appears.



3. Make entries in the following fields:

Field	Description
Control Type	Select IMC S Class 1394/1394 Turbo.
iCODE Version	Select version 3.9 **(**or later).
Interfaces	Select CNET. (The CNET tab displays.)

4. After you complete all other Control Options selections, press **OK**.

Defining Inputs and Outputs

After you have enabled the CNET interface (see *Enabling the CNET Interface in GML Commander*), you need to define:

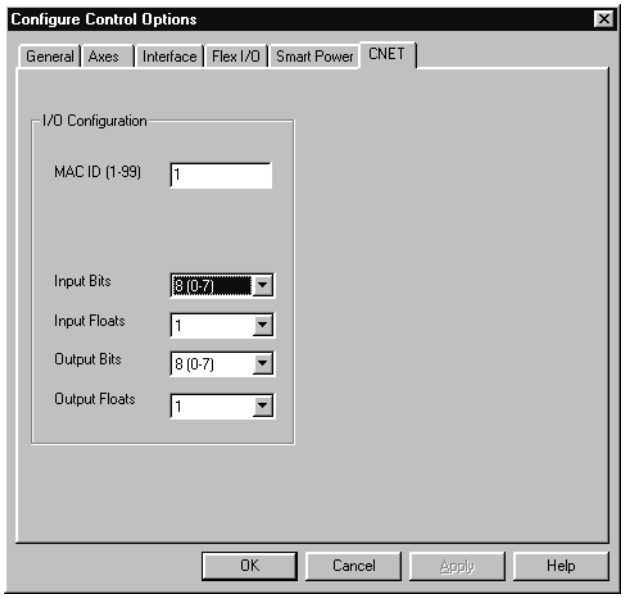
- MAC ID
- Input bits
- Input floats (floating point variables)
- Input group bits
- Output bits
- Output floats (floating point variables)
- Output group bits

Defining the I/O Configuration

To define the I/O Configuration:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select the **CNET** tab. The CNET page appears.

   Note: If no CNET tab appears, on the General page of the Control Options dialog box, select IMC S Class 1394/1394 Turbo in the *Control Type* field, 3.9 (or higher) in the *iCODE Version* field, and the *CNET* field in the *Interface* area (see *Enabling the CNET Interface in GML Commander*). Then select the CNET tab.



4. In the *I/O Configuration* area, make entries in the following fields:

Field	**Description**
MAC ID	Enter a number form 1 to 99
Input Bits[1]	Select the number of input bits you need: 8 (from 0 to 7) 24 (from 0 to 23) 40 (from 0 to 39)
Input Floats[1]	Select the number of input floats you need: 0 to 14 (or 15. See note 1, below.)
Output Bits[1]	Select the number of output bits you need: 8 (from 0 to 7) 24 (from 0 to 23) 40 (from 0 to 39)

Field	Description
Output Floats[1]	Select the number of output floats you need: 0 to 14 (or 15. See note 1, below.)

1 = There are 32 words of I/O available in each CNET I/O file. Each float consists of two 16 bit words. I/O floating point addressing starts at the word following the last discrete I/O.

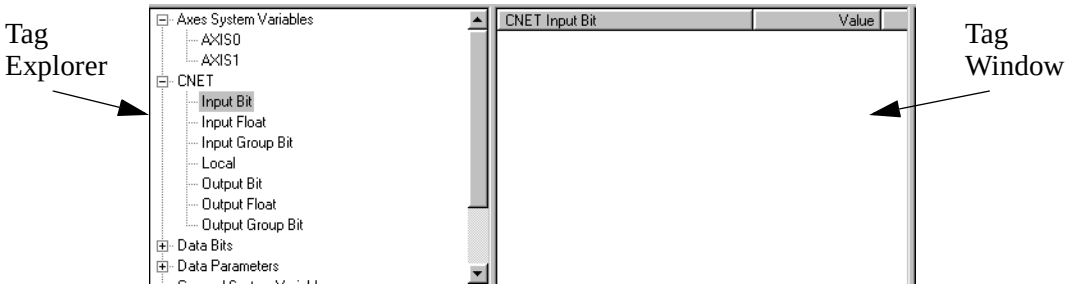
If you:	You can define:
Select eight or 24 discrete I/O points	up to 15 floats.
Configure all 40 I/O points	up to 14 floats.

5. Select **OK**.
6. Define your input bits. See *Defining Input Bits*.

Defining Input Bits

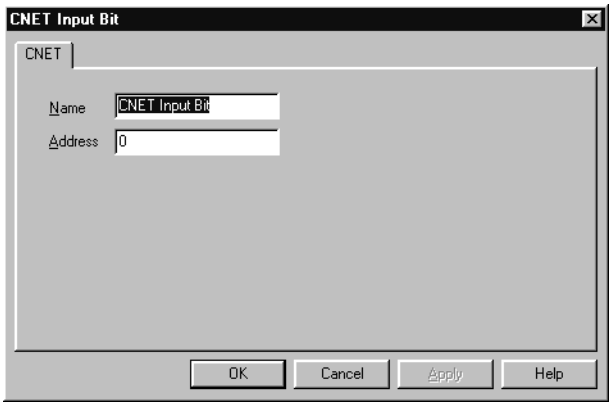
After you have selected the number of input bits required for your application, you can define those bits, either individually or collectively (or both). To define individual input bits:

1. In the Tag Explorer, select **Input Bit**.



2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.

3. Select **New**. The CNET Input Bit dialog box appears.



4. Make entries in the following fields:

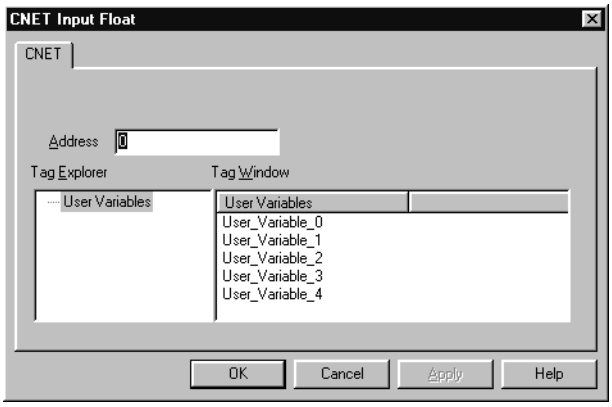
Field	Description
Name	Type the name of the CNET Input Bit.
Address	Type the address, or number, of the bit (from 0 to 39). Note: The available address range depends upon the number of Input Bits you specified in the CNET page of the Configure Control Options dialog box. See *Defining the I/O Configuration.*

5. Select **OK**. The dialog box closes, and the new input appears in the Tag Window.

Defining Input Floats

If you have enabled input floats (see *Defining the I/O Configuration)*, you must link each floating point input to a user variable that holds the floating point value. To do this:

1. In the Tag Explorer, select **Input Float**.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The CNET Input Float dialog box appears.



4. Make entries in the following fields:

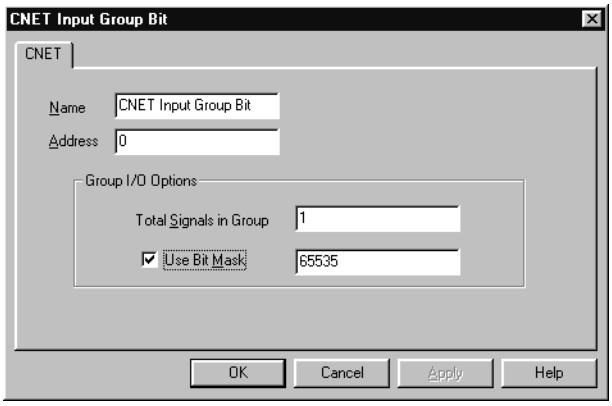
Field	Description
Address	Type the address, or number, of the input float. Note: The available address range depends upon the number of Input Floats (from 0 to 14 or 15) you specified in the CNET page of the Configure Control Options dialog box. See *Defining the I/O Configuration*.
Tag Window	Select a user variable to be linked to the specified input float. Note: You must first define a User Variable if it is to appear in the list. See the *Online Help* for instructions on defining User Variables.

5. Select **OK**. The new Input Float appears in the Tag Window.

Defining Input Group Bits

After you have selected the number of input bits required for your application, you can define those bits, either individually or collectively (or both). To define collections, or groups, of input bits:

1. In the Tag Explorer, select **Input Group Bit.**
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The CNET Input Group Bit dialog box appears.



4. Make entries to the following fields:

Field	Description
Name	Type the name of the CNET Input Bit Group.
Address	Type the address, or number, of the first bit in the group.

Field	Description
Total Signals in Group	Type the number of consecutive bits in the group. Note: This number must include all bits in the sequence—bits whose signals you use, as well as bits in the sequence whose signals you do not use.
Use Bit Mask	Select this if you wish to exclude (mask) one or more signals in the group, then type a binary value representing the bits whose signals are to be read. See the *Optional Mask* section of the *Expression Builder* chapter of this manual for more information on bit masks.

5. Select **OK**. The name of the new CNET Input Bit Group appears in the Tag Window.

Defining Local Variables

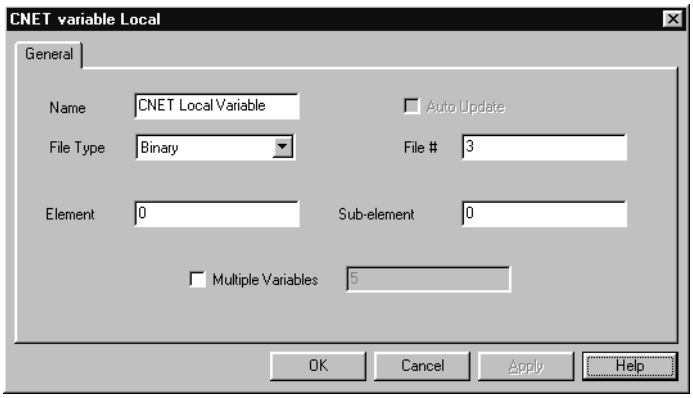
The CNET Local variable stores values in the memory (or data files) of the local motion controller.

CNET and DH-485 Local variables share the same memory space in the local motion controller. Consequently, selecting either DH-485 Local or CNET Local in the Tag Explorer displays the same list of DH-485 and CNET local variables.

To define a CNET Local Variable:

1. In the Tag Explorer, select **Local**.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.

3. Select **New**. The CNET variable Local dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	Type the new variable's name.
File Type	Select a file type: • Binary – Words • Integer– 16-bit Integer Values • Floating – Floating Point Values • ASCII – Characters • BCD – 4-Digit BCD Integers • IntFloat – Floating Point Values
File #	Either accept the file number generated by your selection of a file type, or select a non-pre-defined file type (4, 5, 6, 9, 13, 14, or 15). Note: When you associate a new file number with a file type, that association applies to all later-defined ControlNet and DH-485 local variables in your GML Commander program.

Field	Description
Element	Type the particular element number for the selected File Type. (The available range of available element numbers varies, depending upon the file Type.)
Sub-element	If you selected Binary as the File Type, select a specific sub-element, or bit from 0 to 15.
Multiple Variables	Select this to define more than one variable, then enter the number of variables to be defined. Note: Commander adds a numerical suffix to each variable name, beginning with _0, to distinguish each new variable.
Auto Update	If you selected Floating, you can select Auto Update. The Auto Update tab appears. Note: Use the Auto Update page to let GML Commander assign the new ControlNet variable, during runtime, to an Axis System Variable value or a User Variable. Because these values run to several decimal places, only floating point variables can be used.

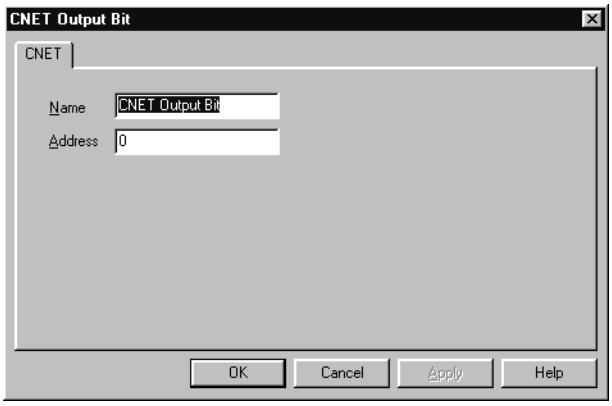
5. Select **OK**. The dialog box closes, and the new input appears in the Tag Window.

Defining Output Bits

After you have selected the number of input bits required for your application, you can define those bits, either individually or collectively (or both). To define individual input bits:

1. In the Tag Explorer, select **Output Bit**.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.

3. Select **New**. The CNET Output Bit dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	Type the name of the CNET Output Bit.
Address	Type the address, or number, of the bit (from 0 to 39). Note: The available address range depends on the number of Output Bits you specified in the CNET page of the Configure Control Options dialog box. See *Defining the I/O Configuration.*

5. Select **OK**. The dialog box closes, and the new input appears in the Tag Window.

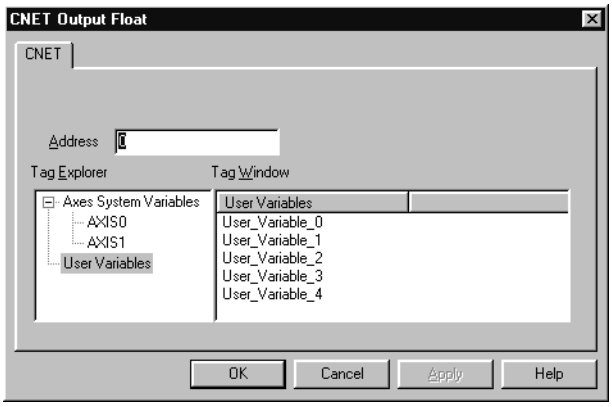
Defining Output Floats

Output floats can pass user variables and a select list of system variables. If you have enabled output floats, you must link each floating point output to a user variable that is the source of the floating-point value. To do this:

1. In the Tag Explorer, select **Output Float**.

2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.

3. Select **New**. The CNET Output Float dialog box appears.



4. Make entries in the following fields:

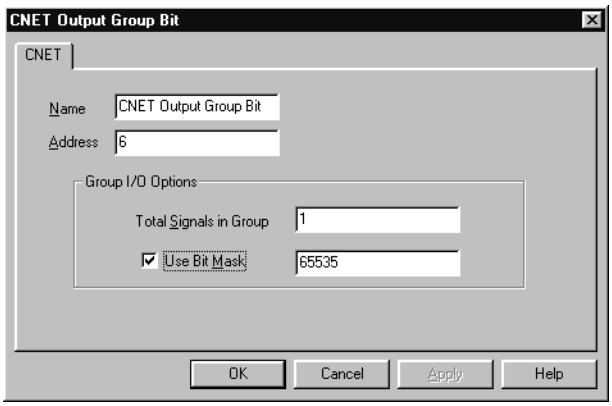
Field	Description
Address	Type the address, or number, of the output float. Note: The available address range depends upon the number of Output Floats (from 0 to 14 or 15) you specified in the CNET page of the Configure Control Options dialog box. See *Defining the I/O Configuration*, above.
Tag Window	Select the user or axis system variable that is the source of the floating point value. Note: You must first define a User Variable if it is to appear in the list. See the *Online Help* for instructions on defining user variables.

5. Select **OK**. The new Output Float appears in the Tag Window.

Defining Output Group Bits

After you have selected the number of output bits required for your application, you can define those bits, either individually or collectively (or both). To define collections, or groups, of input bits:

1. In the Tag Explorer, select **Output Group Bit.**
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The CNET Output Group Bit dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	Type the name of the CNET Output Bit Group.
Address	Type the address, or number, of the first bit in the group.

Field	Description
Total Signals in Group	Type the number of consecutive bits in the group. Note: This number must include all bits in the sequence—bits you will write to, as well as bits in the sequence you will not write to.
Use Bit Mask	To exclude (mask) one or more signals in the group, type a binary value representing the bits whose signals you will write to. See *Bit Masks* in the *Expression Builder* chapter of this manual for more information on bit masks.

5. Select **OK**. The name of the new CNET Output Bit Group appears in the Tag Window.

Editing CNET Bits, Floats, and Variables

The CNET bits, floats, and variables can be easily edited if the need arises. To edit a CNET bit, float, or variable:

1. In the Tag Explorer, under CNET, select the type of bit, float, or variable to edit.
2. In the Tag Window, place the cursor on the specific bit, float, or variable you want to Edit.
3. Click the right mouse button. A pop-up menu appears.
4. Select Edit to open the appropriate CNET dialog box.
5. Edit the settings of the selected bit, float, or variable.
6. Select OK.

Deleting CNET Bits, Floats, and Variables

The CNET bits, floats, and variables can be deleted if they are no longer needed. To delete a CNET bit, float, or variable:

1. In the Tag Explorer, under CNET, select the type of bit, float, or variable to delete.
2. In the Tag Window, place the cursor on the specific bit, float, or variable you want to delete.
3. Click the right mouse button. A pop-up menu appears.
4. Select Delete. The selected bit, float, or variable disappears.

Fault Handling

Refer to the FAULT.GML file on the installation disk for examples of fault routines.

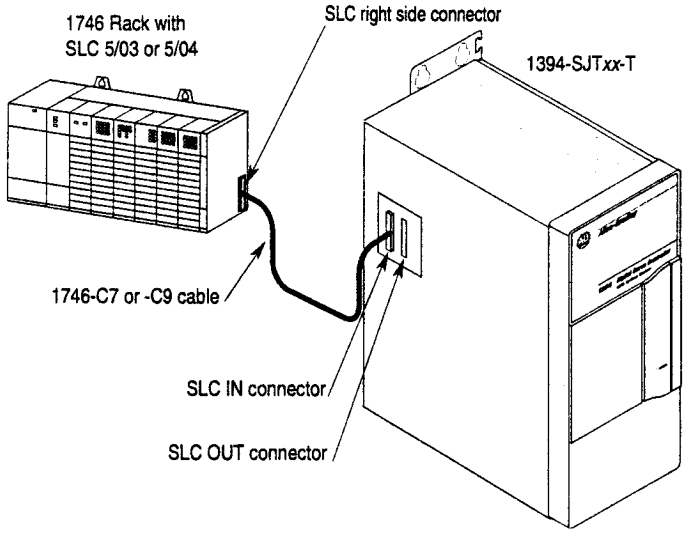
1394 GMC Turbo SLC Interface

You can connect the 1394 GMC Turbo to an SLC rack, containing a 5/03 or 5/04 processor, using a 1746-C7 (flat 6 inch) or 1746-C9 (round 36 inch) SLC local rack extension cable. This connection provides direct backplane communications between the SLC and the 1394.

Note: The SLC 5/03 must be running firmware version OS301 or later. If the SLC interface is enabled, the RIO adapter is not operational.

The interface supports discrete input and output files up to 32 words in length and M0 and M1 files up to 512 words in length. It also supports an ISR interrupt.

Figure 1
Connecting an SLC to a 1394 GMC Turbo



Rack Configurations

The SLC views the 1394 Turbo as a four-slot rack, with one intelligent module installed in the first slot. The 1394 GMC Turbo reserves the remaining empty slots, which cannot be used.

SLC racks are available with 4, 7, 10, and 13 slots. The SLC can only address a maximum of 30 slots and support up to three racks.

The largest logic configuration containing one 1394 GMC Turbo is two 13-slot racks. The 1394 GMC Turbo appears in slot 26. Slots 27, 28 and 29 are reserved.

The largest logic configuration (containing two 1394 GMC Turbos) is one 13-slot rack, in which the first 1394 GMC Turbo appears in slot 13 and the second 1394 GMC Turbo appears in slot 17. Slots 14, 15, 16, 18, 19 and 20 are reserved. Refer to the following figures for more information.

Figure 2
Largest Configuration Containing Two 1394 GMC Turbo Modules

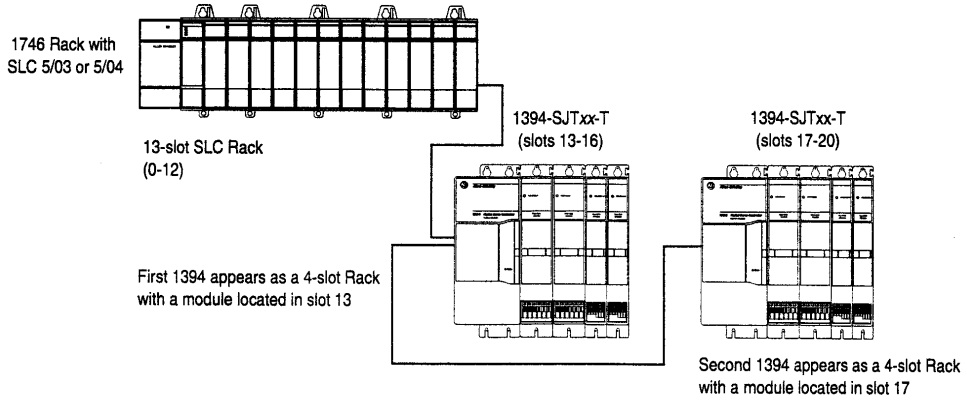
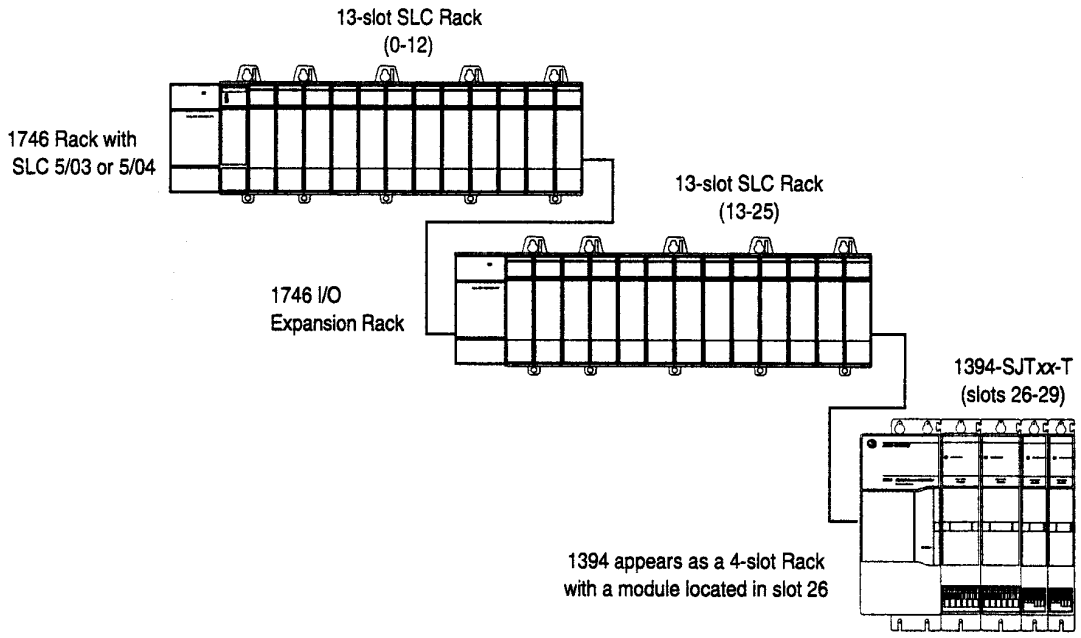


Figure 3
Largest Configuration Containing one 1394 GMC Turbo Module



Selecting the 1394 in Your SLC Software

Before you begin to program the 1394 to communicate with your SLC, you must first select the 1394 as an intelligent module in your SLC programming software.

To select the 1394 as an intelligent module:

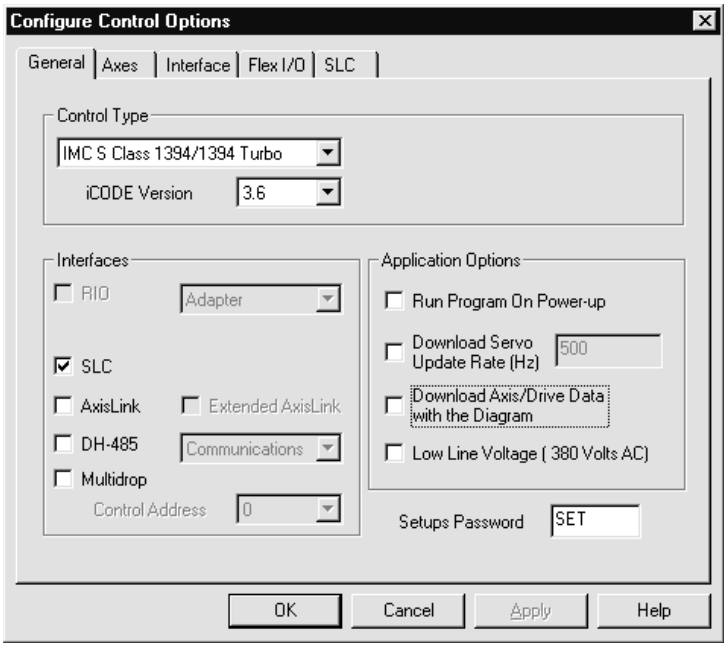
1. Select **Other** from the list of intelligent modules.
2. Type **13617** (the ID code of the 1394) in the *ID Code* field.

Important: The size of the discrete I/O files and the M0 and M1 files impacts both the SLC program scan time and the 1394 GMC Turbo CPU utilization. To optimize your scan time and CPU utilization, be sure to define the size of your discrete I/O and your M0 and M1 files in your SLC program.

Enabling the SLC Interface in GML Commander

To enable the SLC interface in GML Commander:

1. Select **Configure** from the menu bar. The Configure menu displays.
2. Select **Control Options**. The Configure Control Options dialog box appears.



3. Make entries in the following fields:

Field	Description
Control Type	Select IMC S Class 1394/1394 Turbo.
iCODE Version	Select version 3.5 **(**or later).
Interfaces	Select SLC. (The SLC tab appears.) Note: See *Defining the I/O Configuration* and *Enabling M File Transfers* for instructions on completing the SLC page.

4. After you complete all other Control Options selections, press **OK**.

Understanding Discrete Data Transfers

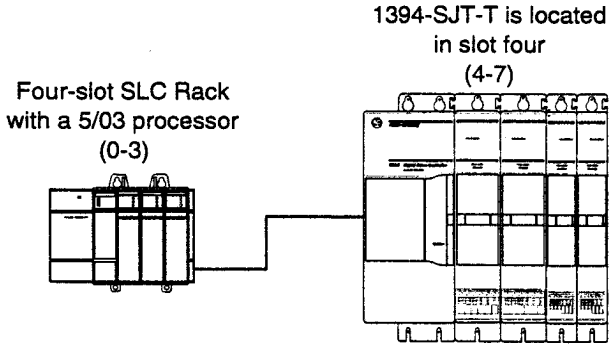
The 1394 GMC Turbo and the SLC use input and output files to transfer discrete data. Each input file and each output file can support up to 32 words (0-31). Those 32 words include:

- Eight dedicated bits and eight user bits (located in word 0).
- Up to 32 additional user bits (located in words 1 and 2).
- Up to 15 floating point variables (located in words 3-31).

Transferring Files from the SLC to the 1394 GMC Turbo

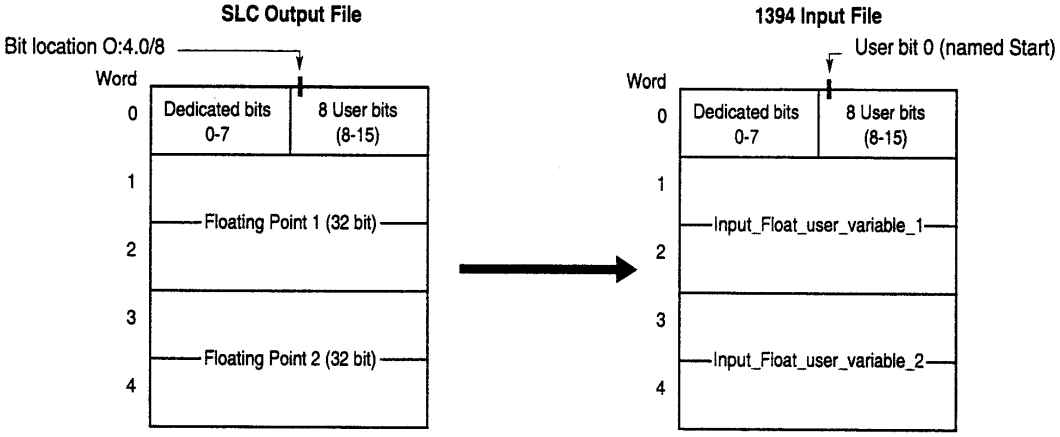
To transfer a discrete file from the SLC to the 1394 GMC Turbo:

1. Determine the slot number of the 1394. In the following figure, the 1394 GMC Turbo appears in slot four.



2. Determine the number of elements to send (0 - 40 user-defined bits, 0 - 15 floating-point variables). In the following figure, eight user-defined bits and two floating-point variables are being transferred.
3. For optimum scan time, define the discrete file size.
4. Define the eight user bits. In the following figure, one of the user bits (O:4.0/8) is named Start (bit 0) in GML Commander.
5. Define the two floating-point variables.
6. Link those floating point variables to GML Commander user variables. In the following figure, the floating-point variables are linked to the GML Commander user variables Input_Float_user_variable_1 and Input_Float_user_variable_2.

Figure 4
Transferring a File from the SLC to the 1394 GMC Turbo

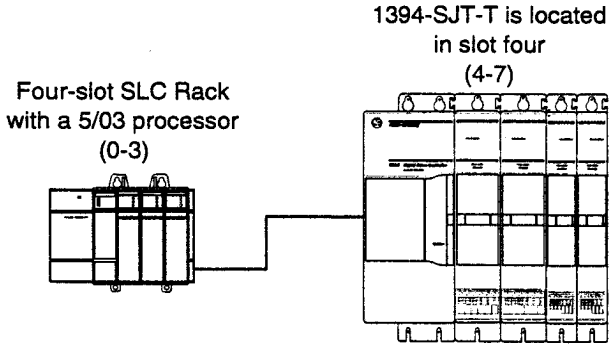


When the SLC finishes its program scan, it updates its outputs. If the GML Commander program is to recognize this new data, the 1394 GMC Turbo must update its inputs. The SLC update rate depends on the size of the user's program. The update rate for the 1394 GMC Turbo depends on the I/O file update rate (see *Defining the I/O Configuration*) and the servo update rate (see *Enabling M File Transfers*). Also, see the *SLC 500 Reference Manual* (Publication 1747-6.15) for specific programming instructions.

Transferring Files from the 1394 GMC Turbo to the SLC

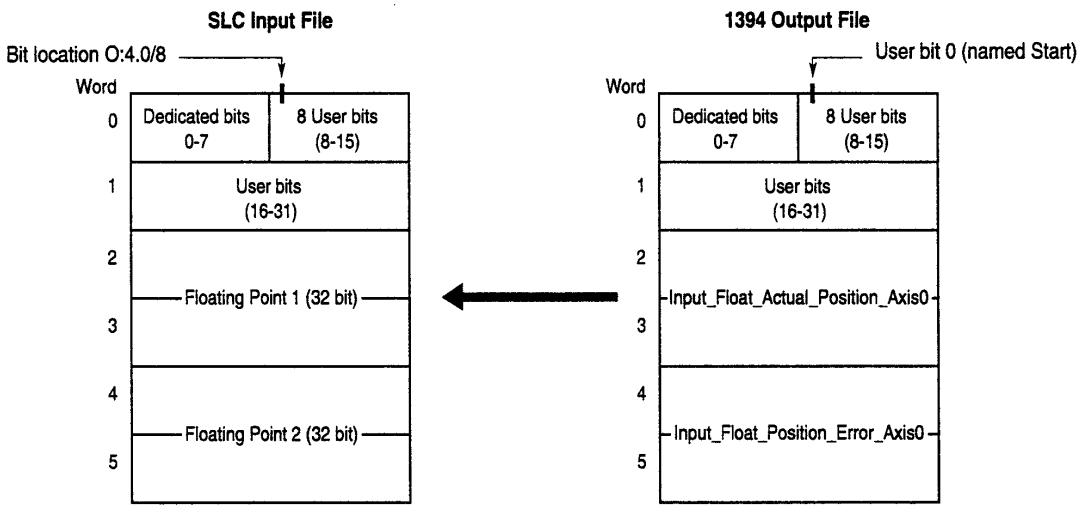
To transfer a discrete file from the 1394 to the SLC:

1. Determine the slot in which the 1394 resides. In the following figure, the 1394 GMC Turbo resides in slot four.



2. Determine the number of elements to send (0 - 40 user-defined bits, 0 - 15 floating-point variables). In the following figure, we are transferring 24 user-defined bits and 2 floating-point variables.
3. For optimum scan time, define the discrete file size.
4. Define the eight user bits. In the following figure, one of the user bits (O:4.0/8) is defined as Start (bit 0) in GML Commander.
5. Define the two floating-point variables. In the following figure, the floating-point variables are linked to the GML Commander user variables *Input_Float_Actual_Position_Axis0* and *Input_Float_Position_Error_Axis0*.

Transferring a File from the 1394 GMC Turbo to the SLC



The rate at which the GML Commander program updates its outputs depends on the I/O file update rate (see *Defining the I/O Configuration*) and the servo update rate (see *Enabling M File Transfers*). In order for the SLC to access this new data, it must update its inputs. The SLC accesses this new data during the next input scan. The rate at which the SLC program updates its inputs depends on the size of the user's program.

See the *SLC 500 Reference Manual* (publication 1747-6.15) for specific programming instructions.

Dedicated/User-Defined I/O Bits

The first word in each I/O file contains dedicated Control/Status bits (Bits 0-7) and user-defined I/O bits (Bits 8-15). The minimum SLC input bit configuration consists of one 16-bit word. The maximum input bit configuration consists of three 16-bit words. The following tables present examples of minimum and maximum input bit configurations, and definitions of the dedicated Control/Status bits.

Table 1: 1394 GMC Turbo Input Data from the SLC (minimum configuration, one 16-bit word)

Word	Bit	Description	Definition
0	0/0	Run program	When this bit changes from 0 to 1, the GML Commander application program executes from the beginning. If the application program is already running or paused, the change has no effect. The SLC stop bit described below has priority over this bit. The application program cannot access this bit.
	0/1	Stop program	When this bit is set to 1, the application program stops. The application program cannot access this bit. This bit has priority over the Run and Pause bits.
	0/2	Pause/resume program	When this bit changes from 0 to 1, the GML Commander application program pauses. When the bit changes from 1 to 0, the program resumes. The application program cannot access this bit.
	0/3	SLC process fault	When the SLC sets this bit to 1, the 1394 GMC Turbo system software generates a global fault. The corresponding global fault code indicates the SLC process fault. The application program cannot access this bit.
	0/4	Reserved	N/A
	0/5	Reserved	N/A
	0/6	M0 File update request	If M file manual is enabled, the SLC ladder program can set this bit to 1, which causes the 1394 GMC Turbo to update the M0 file. The application program references this bit as the system variable SLC_M0_Update_Request.
	0/7	M1 file update acknowledge	If M file manual is enabled, the SLC ladder program can set this bit to 1, which causes the SLC ladder program to acknowledge the 1394's request to update the M1 file. The application program references this bit as the system variable SLC_M1_Update_Acknowledge.
	0/8 – 0/15	User defined	N/A

Table 2: 1394 GMC Turbo Input Data from the SLC (maximum configuration, three 16-bit words)

Word	Bit	Description	Definition
1	0/0	Run program	When this bit changes from 0 to 1, the GML Commander application program executes from the beginning. If the application program is already running or paused, the change has no effect. The SLC stop bit described below has priority over this bit. The application program cannot access this bit.
	0/1	Stop program	When this bit is set to 1, the application program stops. The application program cannot access this bit. This bit has priority over the Run and Pause bits.
	0/2	Pause/resume program	When this bit changes from 0 to 1, the GML Commander application program pauses. When the bit changes from 1 to 0, the program resumes. The application program cannot access this bit.
	0/3	SLC process fault	When the SLC sets this bit to 1, the 1394 GMC Turbo system software generates a global fault. The corresponding global fault code indicates the SLC process fault. The application program cannot access this bit.
	0/4	Reserved	N/A
	0/5	Reserved	N/A
	0/6	M0 File update request	If M file manual is enabled, the SLC ladder program can set this bit to 1, which causes the 1394 GMC Turbo to update the M0 file. The GML Commander program references this bit as the system variable SLC_M0_Update_Request.
	0/7	M1 file update acknowledge	If M file manual is enabled, the SLC ladder program can set this bit to 1, which causes the SLC ladder program to acknowledge the 1394's request to update the M1 file. The GML Commander program references this bit as the system variable SLC_M1_Update_Acknowledge.
	0/8 – 0/15	User defined	N/A
1	1/0 – 1/15	User defined	N/A
2	2/0 – 2/15	User defined	N/A

Table 3: 1394 GMC Turbo Output Data from the SLC (minimum configuration, one 16-bit word)

Word	Bit	Description	Definition
0	0/0	Program running	This bit is set to 1 by the 1394 GMC Turbo while the GML Commander application program is running. This bit cannot be accessed directly by the GML Commander program.
	0/1	Global fault	The 1394 GMC Turbo sets this bit to 1 if a fault has occurred. This bit cannot be accessed directly by the GML Commander application program.
	0/2	Reserved	N/A
	0/3	Reserved	N/A
	0/4	Reserved	N/A
	0/5	M file handshake enabled	When set to 1, this bit indicates that M files will be updated using manual mode. When set to 0, this bit indicates that M files will be updated using automatic mode.
	0/6	M0 File update acknowledge	This bit is set to 1 by the GML Commander application program to indicate that an M0 file update has been acknowledged and in process. Bit 6 can be written by the application program. This bit is referenced by the application program as the system variable SLC_M0_Update_Acknowledge.
	0/7	M1 file update request	This bit is set to 1 by the GML Commander application program to request an M1 file update. The 1394 GMC Turbo system software resets this bit to 0 when the update is complete. Bit 7 can be written by the application program. This bit is referenced by the application program as the system variable SLC_M1_Update_Request.
	0/8 – 0/15	User defined	N/A

Table 4: 1394 GMC Turbo Output Data from the SLC (maximum configuration, three 16-bit words)

Word	Bit	Description	Definition
0	0/0	Program running	This bit is set to 1 by the 1394 GMC Turbo while the GML Commander application program is running. This bit cannot be accessed directly by the application program.
	0/1	Global fault	The 1394 GMC Turbo sets this bit to 1 if a fault has occurred. This bit cannot be accessed directly by the GML Commander application program.
	0/2	Reserved	N/A
	0/3	Reserved	N/A
	0/4	Reserved	N/A
	0/5	M file handshake enabled	When set to 1, this bit indicates that M files will be updated using manual mode. When set to 0, this bit indicates that M files will be updated using automatic mode.
	0/6	M0 File update acknowledge	This bit is set to 1 by the GML Commander application program to indicate that an M0 file update has been acknowledged and in process. Bit 6 can be written by the application program. The application program refers to this bit as the system variable SLC_M0_Update_Acknowledge.
	0/7	M1 file update request	This bit is set to 1 by the GML Commander application program to request an M1 file update. The 1394 GMC Turbo system software resets this bit to 0 when the update is complete. Bit 7 can be written by the application program. The application program as the system variable SLC_M1_Update_Request.
	0/8 – 0/15	User defined	N/A
1	1/0 – 1/15	User defined	N/A
2	2/0 – 2/15	User defined	N/A

Defining Inputs and Outputs

After you have enabled the SLC interface (see *Enabling the SLC Interface in GML Commander*), you need to define:

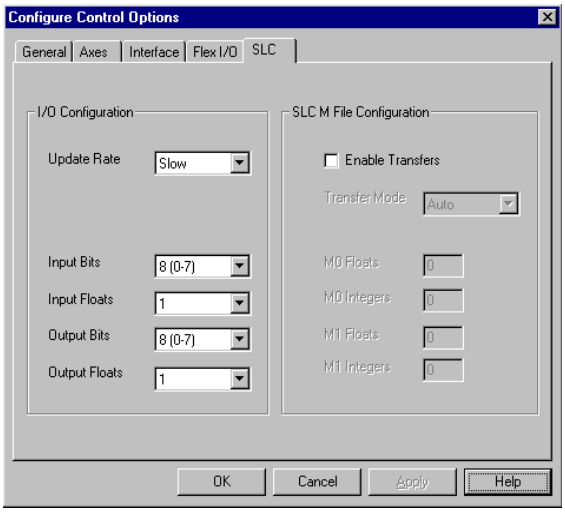
- I/O configuration
- Input bits
- Input floats (floating point variables)
- Input group bits
- Output bits
- Output floats (floating point variables)
- Output group bits

Defining the I/O Configuration

To define the I/O Configuration:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select the **SLC** tab. The SLC page appears.

   Note: If no SLC tab appears, on the General page of the Control Options dialog box, select IMC S Class 1394/1394 Turbo in the *Control Type* field, 3.5 (or higher) in the *iCODE Version* field, and the *SLC* field in the *Interface* area (see *Enabling the SLC Interface in GML Commander*). Then select the SLC tab.



4. In the *I/O Configuration* area, make entries in the following fields:

Field	**Description**
Update Rate	I/O image file updates to and from the SLC are executed from the servo interrupt routine, and are independent of the SLC I/O scan. The higher the setting, the more CPU utilization is impacted. Select one of the I/O update rates: Slow — Updates the complete I/O image file every four servo loops. Medium — Updates the complete I/O image file every two servo loops. Fast — Updates the complete I/O image file every servo interrupt.
Input Bits[1]	Select the number of input bits you need: 8 (from 0 to 7) 24 (from 0 to 23) 40 (from 0 to 39)

Field	Description
Input Floats[1]	Select the number of input floats you need: 0 to 14 (or 15. See note 1, below.)
Output Bits[1]	Select the number of output bits you need: 8 (from 0 to 7) 24 (from 0 to 23) 40 (from 0 to 39)
Output Floats[1]	Select the number of output floats you need: 0 to 14 (or 15. See note 1, below.)

1 = There are 32 words of I/O available in each SLC I/O file. Each float consists of two 16 bit words. I/O floating point addressing starts at the word following the last discrete I/O.

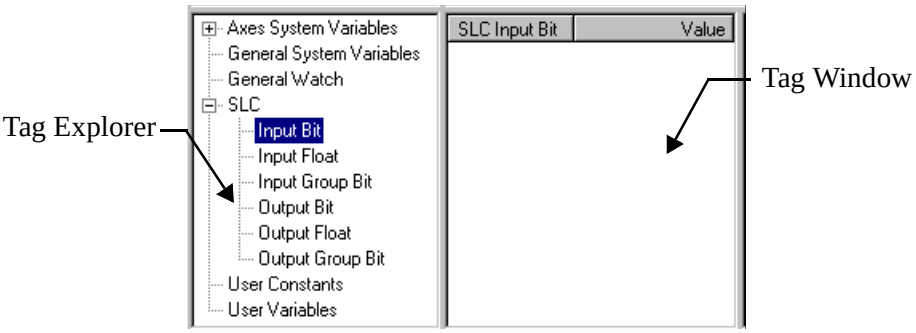
If you:	You can define:
Select eight or 24 discrete I/O points	up to 15 floats.
Configure all 40 I/O points	up to 14 floats.

5. Select **OK**.
6. Define your input bits. See *Defining Input Bits*.

Defining Input Bits

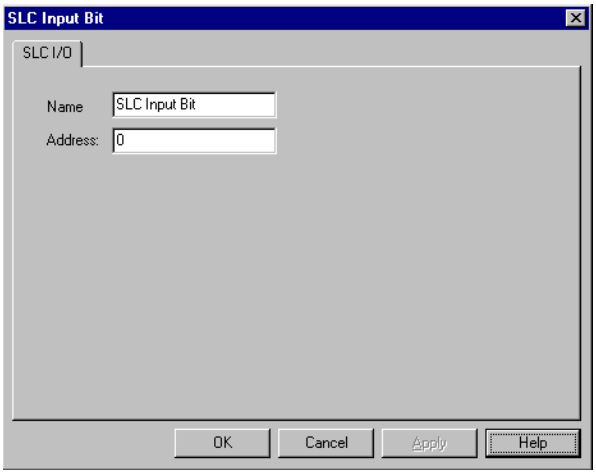
After you have selected the number of input bits required for your application, you can define those bits, either individually or collectively (or both). To define individual input bits:

1. In the Tag Explorer, select **Input Bit**.



2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.

3. Select **New**. The SLC Input Bit dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	Type the name of the SLC Input Bit.

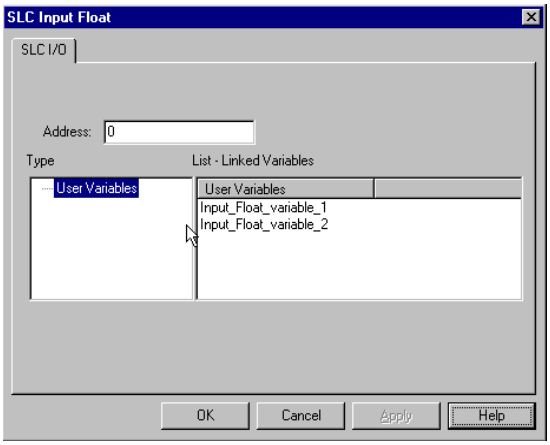
Field	Description
Address	Type the address, or number, of the bit (from 0 to 39). Note: The available address range depends upon the number of Input Bits you specified in the SLC page of the Configure Control Options dialog box. See *Defining the I/O Configuration*.

5. Select **OK**. The dialog box closes, and the new input appears in the Tag Window.

Defining Input Floats

If you have enabled input floats (see *Defining the I/O Configuration)*, you must link each floating point input to a user variable that holds the floating point value. To do this:

1. In the Tag Explorer, select **Input Float**.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The SLC Input Float dialog box appears.



4. Make entries in the following fields:

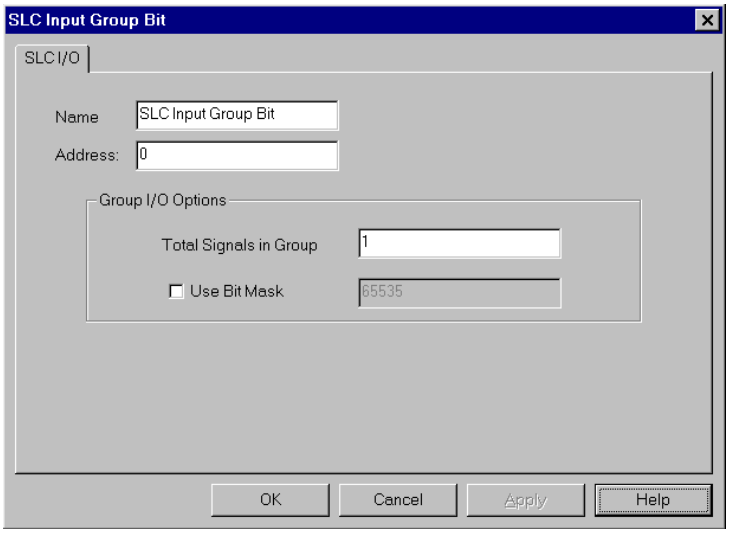
Field	Description
Address	Type the address, or number, of the input float. Note: The available address range depends upon the number of Input Floats (from 0 to 14 or 15) you specified in the SLC page of the Configure Control Options dialog box. See *Defining the I/O Configuration*.
List - Linked Variables	Select a user variable to be linked to the specified input float. Note: You must first define a User Variable if it is to appear in the list. See the *Online Help* for instructions on defining User Variables.

5. Select **OK**. The new Input Float appears in the Tag Window.

Defining Input Group Bits

After you have selected the number of input bits required for your application, you can define those bits, either individually or collectively (or both). To define collections, or groups, of input bits:

1. In the Tag Explorer, select **Input Group Bit.**
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The SLC Input Group Bit dialog box appears.



4. Make entries to the following fields:

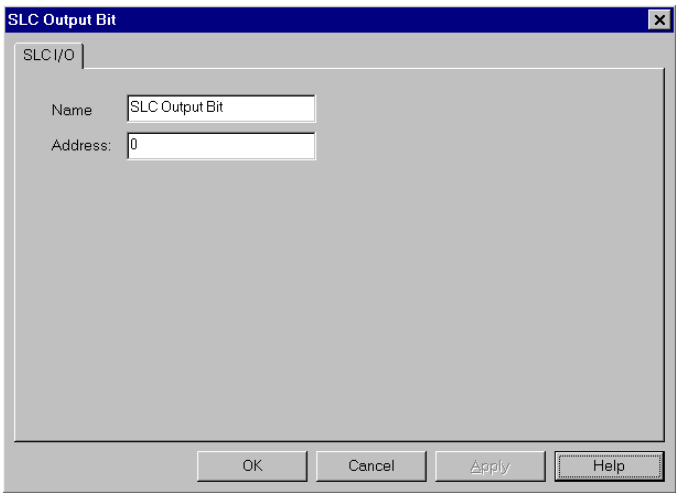
Field	Description
Name	Type the name of the SLC Input Bit Group.
Address	Type the address, or number, of the first bit in the group.
Total Signals in Group	Type the number of consecutive bits in the group. Note: This number must include all bits in the sequence—bits whose signals you use, as well as bits in the sequence whose signals you do not use.
Use Bit Mask	Select this if you wish to exclude (mask) one or more signals in the group, then type a binary value representing the bits whose signals are to be read. See the *Optional Mask* section of the *Expression Builder* chapter of this manual for more information on bit masks.

5. Select **OK**. The name of the new SLC Input Bit Group appears in the Tag Window.

Defining Output Bits

After you have selected the number of input bits required for your application, you can define those bits, either individually or collectively (or both). To define individual input bits:

1. In the Tag Explorer, select **Output Bit**.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The SLC Output Bit dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	Type the name of the SLC Output Bit.

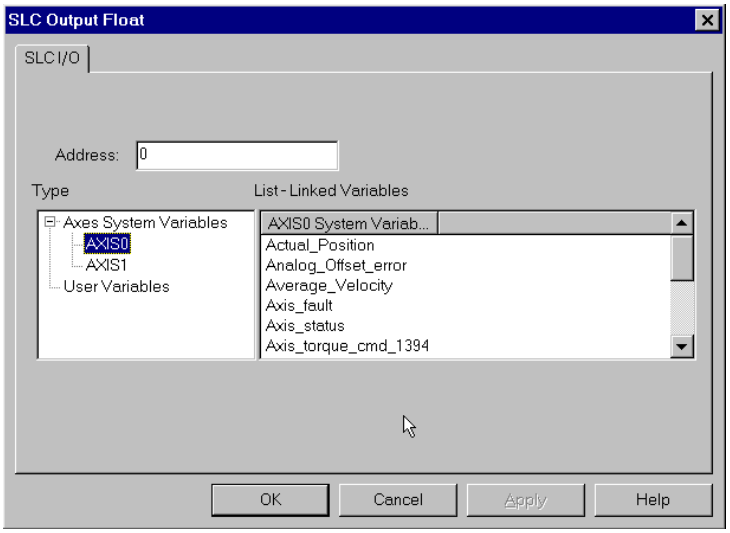
Field	Description
Address	Type the address, or number, of the bit (from 0 to 39). Note: The available address range depends on the number of Output Bits you specified in the SLC page of the Configure Control Options dialog box. See *Defining the I/O Configuration*.

5. Select **OK**. The dialog box closes, and the new input appears in the Tag Window.

Defining Output Floats

Output floats can pass user variables and a select list of system variables. If you have enabled output floats, you must link each floating point output to a user variable that is the source of the floating-point value. To do this:

1. In the Tag Explorer, select **Output Float**.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The SLC Output Float dialog box appears.



4. Make entries in the following fields:

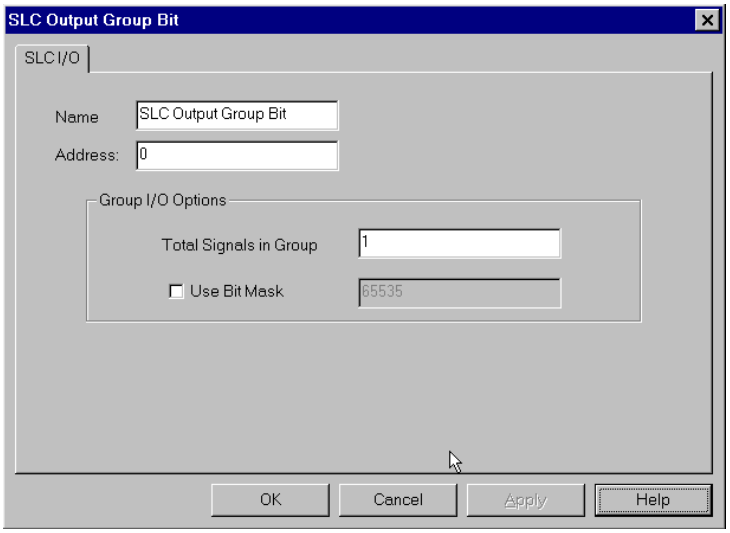
Field	Description
Address	Type the address, or number, of the output float. Note: The available address range depends upon the number of Output Floats (from 0 to 14 or 15) you specified in the SLC page of the Configure Control Options dialog box. See *Defining the I/O Configuration*, above.
List	Select the user or axis system variable that is the source of the floating point value. Note: You must first define a User Variable if it is to appear in the list. See the *Online Help* for instructions on defining user variables.

5. Select **OK**. The new Output Float appears in the Tag Window.

Defining Output Group Bits

After you have selected the number of output bits required for your application, you can define those bits, either individually or collectively (or both). To define collections, or groups, of input bits:

1. In the Tag Explorer, select **Output Group Bit.**
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The SLC Output Group Bit dialog box appears.



4. Make entries in the following fields:

Field	Description
Name	Type the name of the SLC Output Bit Group.
Address	Type the address, or number, of the first bit in the group.

Field	Description
Total Signals in Group	Type the number of consecutive bits in the group. Note: This number must include all bits in the sequence—bits you will write to, as well as bits in the sequence you will not write to.
Use Bit Mask	To exclude (mask) one or more signals in the group, type a binary value representing the bits whose signals you will write to. See *Bit Masks* in the *Expression Builder* chapter of this manual for more information on bit masks.

5. Select **OK**. The name of the new SLC Output Bit Group appears in the Tag Window.

Understanding M Files

M files transfer large, non-time-critical groups of data—such as endpoints and CAM tables—between the 1394 GMC Turbo and the SLC. M files are updated whenever they are accessed during read/write functions. Each M file is 1024 bytes long and can consist of one of the following:

- All integers.
- All floating points.
- A combination of both.

If an M File contains all:	The maximum number of elements is:
Integer elements	512 (each integer element is 16 bits (one word)).
Floating point elements	256 (each floating point element is 32 bits (two words)).

If an M file contains both integers and floating points, the floating point elements are located in front of the integer elements. Use the formula below to determine the proportions of your combination:

(number of floating points x 2) + (number of integers) ≤ 512

For example: If your M file contains 100 floating-point elements, the maximum number of integer elements available is 312.

M files can be accessed using indirect variables. This allows elements in the M file to be indexed like an array. M file variables do not need assigned names to be referenced indirectly.

Types of M Files

There are two types of M Files:

- M1 files are used to transfer data from the 1394 GMC Turbo to the SLC.
- M0 files are used to transfer data from the SLC to the 1394 GMC Turbo.

Transfer Modes

GML Commander supports two different M file update modes:

- Automatic (auto) mode.
- Handshaking (manual) mode.

Auto Mode

When auto mode is enabled, the SLC notifies the 1394 GMC Turbo every time it writes new data to the M0 file. The 1394 GMC Turbo does not know which portion of the M0 file has been updated, and therefore copies the entire file each time—even if only one element was written. Depending on your application and the size of your M files, these numerous updates can impact your CPU utilization. Auto mode is ideal for applications with small M files containing elements that do not have to be updated simultaneously.

Manual Mode

When manual mode is enabled, the 1394 GMC Turbo ignores the automatic SLC update notifications and waits for the program to set a system bit. This decreases the number of updates, and therefore minimizes the impact on your CPU utilization. Manual mode is ideal for applications with large M files containing elements that need to be updated simultaneously.

For example: If your application requires position, velocity and acceleration data to make a move, and those elements are located in a file with 100 other elements, you must update all of the elements at the same to make sure that the data you need is current.

Using auto mode, you cannot verify that the three elements you need were updated at the same time because they may have been updated at different points during the program.

Using manual mode, you can update all of the elements you need and then transfer them all at the same time instead of updating each element one at a time.

Updating M0 Files Using Manual Mode

When M0 files are updated using manual mode:

- The SLC program updates the M0 file variables and issues the M0 Update Request bit (bit 6).
- The GML Commander program sees the system bit variable SLC_M0_Update_Request (bit 6) and sets the system bit variable SLC_M0_Update_Acknowledge.
- The 1394 GMC Turbo system software waits for the GML Commander program to set this bit and then starts the update.
- The SLC program waits for the M0 Update Acknowledge bit to be set and then resets the M0 Update Request bit. This serves as a signal to the 1394 GMC Turbo system that the SLC has recognized the start of the update.

- After the 1394 GMC Turbo completes the update, and the M0 Update Request bit has been reset, the 1394 system software resets the system bit variable SLC_M0_Update_Acknowledge.
- The GML Commander program waits for the M0_Update_Acknowledge bit to reset and the update is complete.

Figure 6: Typical Ladder Logic Diagram for M0 File Transfers

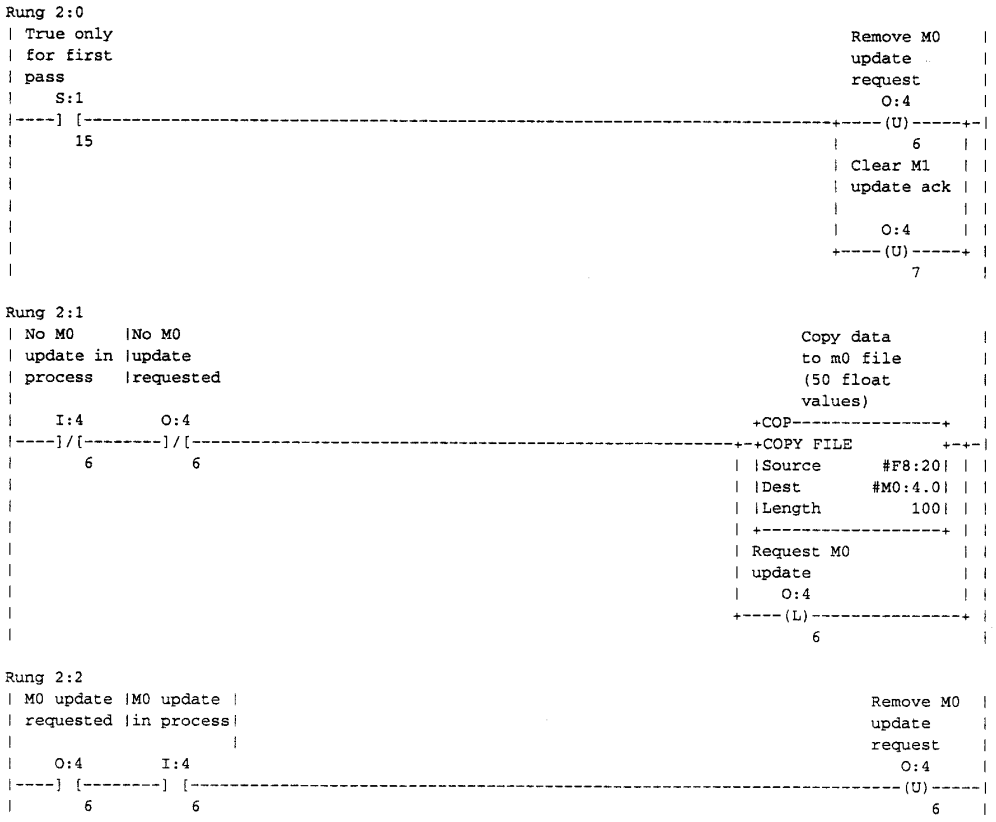
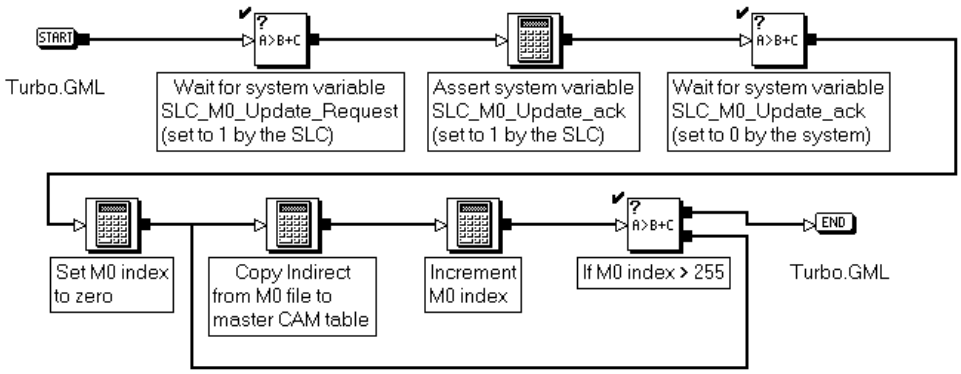


Figure 7: Typical GML Commander Program for M0 File Transfers



Updating M1 Files using Manual Mode

When M1 files are updated using manual mode:

- The GML Commander program updates the M1 variables, and then issues the system bit variable SLC_M1_Update_Request.
- Once the SLC program sees the request, it issues the M1 Update Acknowledge bit when ready.
- Once both bits are issued, the 1394 GMC Turbo system software begins the update.
- When the update is completed, the 1394 GMC Turbo resets the SLC_M1_Update_Request.
- The SLC waits for the M1 Update Request bit to reset and then clears the M1 Update Acknowledge bit. The update is completed and the SLC can copy the new data from the M1 file.

Figure 8: Typical Ladder Logic Diagram for M1 File Transfers

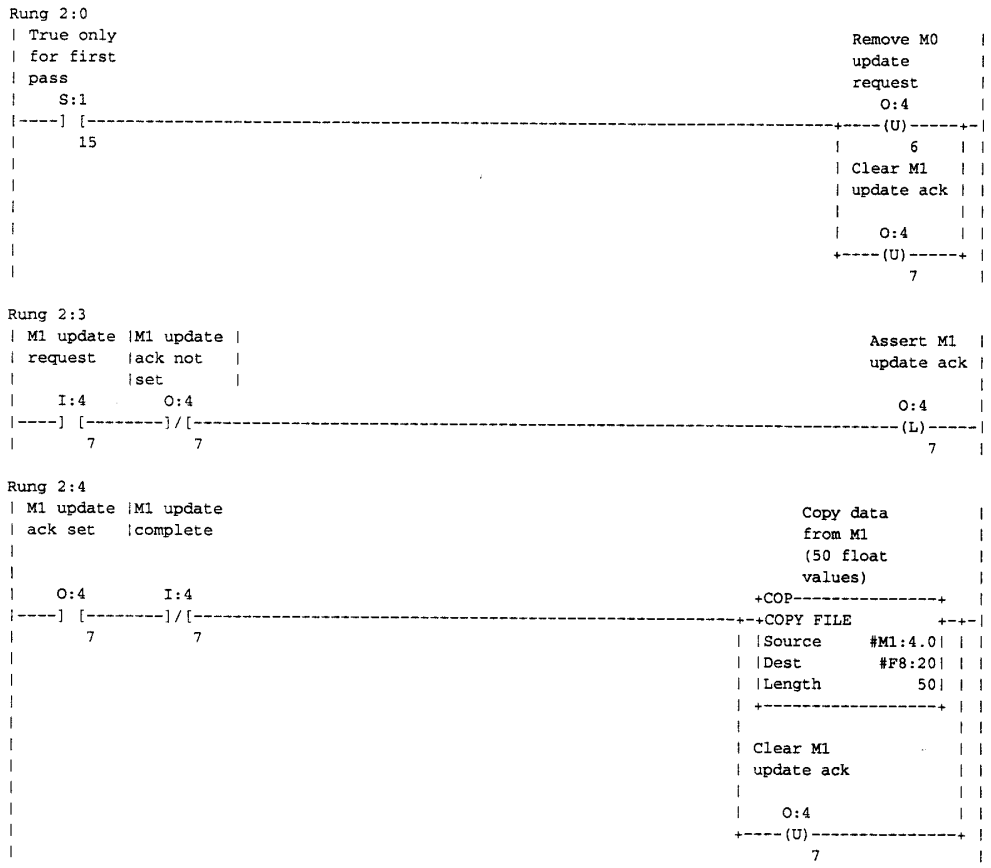
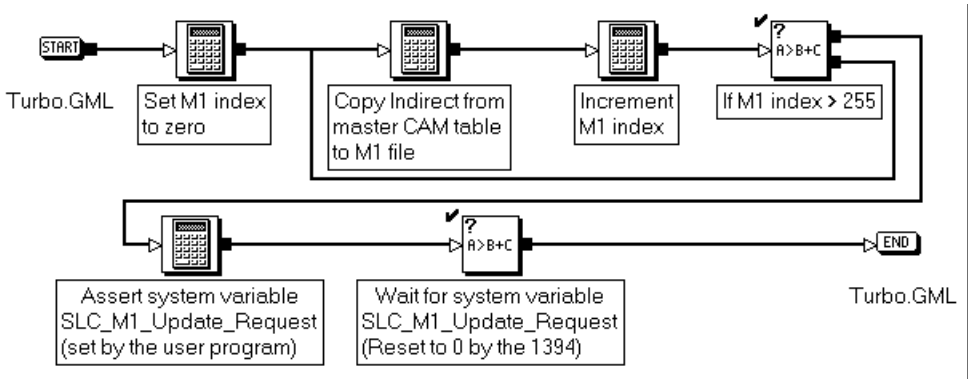


Figure 9: Typical GML Commander Program for M1 File Transfers

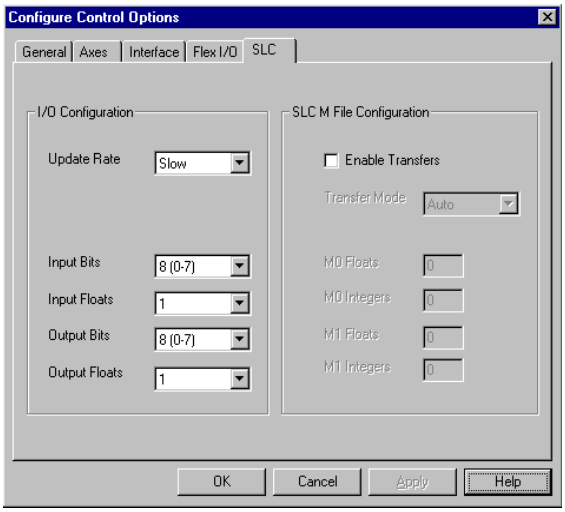


Enabling M File Transfers

To enable M file transfers in GML Commander:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select the **SLC** tab. The SLC page appears.

   Note: If the SLC tab doesn't appear, on the General page of the Control Options dialog box, select IMC S Class 1394/1394 Turbo in the *Control Type* field, 3.5 (or higher) in the *iCODE Version* field, and the *SLC* field in the *Interface* area (see *Enabling the SLC Interface in GML Commander*). Then select the SLC tab.



4. Complete SLC M File Configuration portion of the dialog box, by making the following entries:

Field	Description
Enable Transfers	Select this checkbox to enable transfers.
Transfer Mode	Select a transfer mode. Refer to the *Transfer mode* section for more information.
M0 Floats	Type the number of floats you need in the M0 file.
M0 Integers	Type the number of integers you need in the M0 file.
M1 Floats	Type the number of floats you need in the M1 file.
M1 Integers	Type the number of integers you need in the M1 file.

Note: The number of M0 and M1 words specified above should match the number of words set in the SPIO section of your SLC 500 program software.

5. Select **OK**.
6. Define your M0 Floats. See *Defining M0 Floats*.

Defining M0 and M1 Floats and Integers

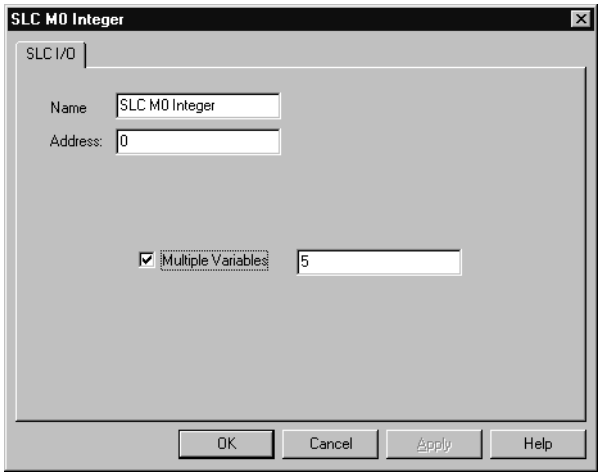
The procedures for defining M0 Floats, M0 Integers, M1 Floats and M1 Integers are virtually identical. In each case, you create a variable—or an array of variables— using the same dialog box (except for the dialog box's name).

But, as you define these variables, remember:

- M1 Floats and M1 Integers transfer data from the 1394 GMC Turbo to the SLC.
- M0 Floats and M0 Integers transfer data from the SLC to the 1394 GMC Turbo.

To define M0 and M1 floats and integers:

1. In the Tag Explorer, select the type of variable.
2. In the Tag Window, single-click the right mouse button. A variable short-cut menu appears.
3. Select **New**. The appropriate dialog box appears. For example, the SLC M0 Float dialog box appears below.



4. Make entries to the following fields:

Field	Description
Name	Type the name of the float or integer.
Address	Type the address, or number, of the float or integer. Note: If you also select Multiple Variables, the address you type is the address of the first variable in the array.
Multiple Variable(s)	Select this checkbox to create a variable array. Then, type in the number of variables to include in the variable array.

5. Select **OK**. The variable, or array of variables, you have just defined appear in the Tag Window.

Fault Handling

Refer to the FAULT.GML file on the installation disk for examples of fault routines.

Interrupts

The Interrupt SLC instruction block allows the 1394 to initiate a ladder interrupt routine in the SLC. You can use this command as an action in an Input Event Action statement, or include it directly in the GML Commander program.

SLC inputs have also been added to the Input Event Action block, which means you now can generate an event (interrupt) using SLC inputs.

Refer to the *Input* section of the *Function Blocks* chapter of this manual, and your *SLC 500 Reference* manual (publication 1747-6.15) for more information.

Programmable Limit Switch Example

The purpose of this example is to show how to transfer a floating point variable to the SLC using an SLC input file.

Equipment Configuration	The equipment for this example consists of: • A four-slot rack with an SLC 5/03 processor. • An SLC output module located in slot one. • A 1394 GMC Turbo located in slot four.
Task	To set an output in the SLC based on the position of Axis 0, that is controlled by the 1394 GMC Turbo.
Solution	Send the axis position to the SLC. Once the SLC receives the Axis 0 position, it evaluates that position and then sets an output.
Programming	Use discrete file transfer to send the Axis 0 position to the SLC as a floating-point value.

To transfer a floating point variable to the SLC using an SLC input file, you must do the steps shown in the following sections:

- GML Commander program
- SLC program

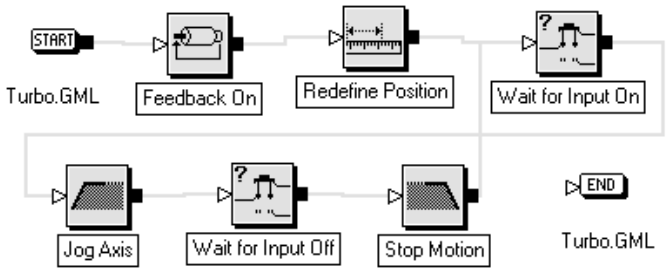
GML Commander Program

The purpose of the following GML Commander program, is to move Axis 0 so that the system variable *Actual Position* changes.

In your GML Commander program:

- Define the discrete file size.
- Define the number of floats. For the program below we need one output float and no input floats.
- Link the output float to the system variable *Actual Position* for Axis 0.

Figure 10: GML Commander Program for a Limit Switch

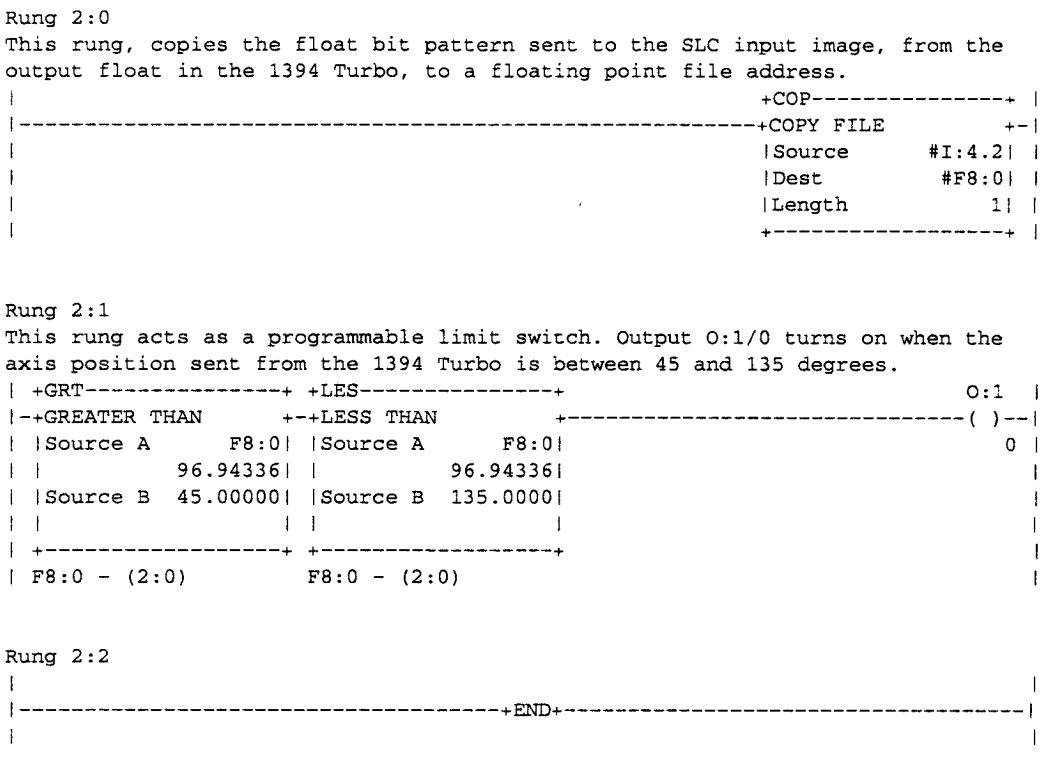


SLC Program

In your SLC program:

- Determine which slot the 1394 resides in.
- Move the data from the input file to the floating-point file.
- Use the floating-point file in a comparison to determine the state of the output bit (1 or 0).

Figure 11: SLC Ladder Logic Diagram for a Programmable Limit Switch



Initializing Your Motion Controller

There are two types of initialization for your motion controllers:

- Hardware initialization.
- Software initialization.

Although both actions initialize or reset certain aspects of the motion controller, their use and effect is very different.

Initializing the Hardware

Hardware initialization (also called INIT) restores communication with the motion controller if a communication parameter becomes corrupted, or if the serial ports no longer appear to be functioning properly even though the controller powers-up properly. If resetting the controller does not restore proper operation, perform hardware initialization.

ATTENTION: Do not hardware initialize a motion controller unless you have uploaded the application program and setup data files and saved them on your computer.

1394 GMC and 1394 GMC Turbo System Module

To hardware initialize a 1394 GMC or 1394 GMC Turbo System Module motion controller (1394-SJTxx models), do the following:

1. With power on, open the front cover of the system module. Unlock the controller's memory by turning the front-panel keyswitch to the unlocked position.
2. Press in and hold the INIT button on the power supply.

3. Press and release the RESET button on the system module. (Do not release the INIT for at least 10 seconds or until the System OK LED glows steadily.)
4. Release the INIT button.
5. Press and release the RESET button again.
6. Close the cover.

Compact

To hardware initialize a Compact motion controller (IMC-S/23x models), do the following:

1. Unlock the controller's memory by turning the front-panel keyswitch to the unlocked position.
2. Remove the key.
3. With power applied, open the front cover.
4. Press in and hold the INIT button on the power supply module.
5. Press and release the RESET button on the front panel of the motion controller module. (Do not release the INIT button until the green System OK LED glows steadily.)
6. Release the INIT button.
7. Press and release the RESET button again.
8. Close the cover.

Integrated

To hardware initialize an Integrated motion controller (IMC-S/21x models), do the following:

1. Unlock the controller's memory by turning the front-panel keyswitch to the UNLOCKED position.
2. With power applied, install a temporary INIT jumper between TB2-4 to TB2-5.

3. Press and release the front panel RESET button.
4. Remove the temporary INIT jumper.
5. Press and release the front panel RESET button again.

Basic

To hardware initialize a Basic motion controller (IMC-S/20x models), do the following:

1. Unlock the controller's memory by installing a memory unlock jumper between TB2-2 and TB2-4.
2. With power applied, install a temporary INIT jumper between TB2-1 to TB2-4.
3. Press and release the front panel RESET button.
4. Remove the temporary INIT jumper.
5. Press and release the front panel RESET button again.

Effect of Hardware Initialization

When the motion controller is hardware initialized, the following occurs:

- The baud rate for both serial ports (usually entered in the controller's application setup menu) is set to 9600. See the *Setup and Installation Manual* for your controller.
- The runtime display and operator interface ports (usually set in the interface page of the Configure Control Options dialog box) are set to Port B.
- Direct mode communication is reset for full duplex communication and to generate linefeeds and carriage returns.
- The setup menu password (usually entered in the General page of the Configure Control Options dialog box) is reset to SET.

- The controller is set to not automatically run program on power-up (usually entered in the General page of the Configure Control Options dialog box).
- Multidrop operation (usually set in the General page of the Configure Control Options dialog box) is disabled.
- Remote I/O, AxisLink, and DH-485 communication (usually set in the General page of the Configure Control Options dialog box) are disabled.
- Status 0 LED glows a continuous, uninterrupted red.

After communication is restored or the problem is fixed, you must set the above parameters back to their appropriate values for your application. You can do this manually from the motion controller's built-in setup menus, or by downloading the application program with *Download Axis/ Drive Data with the Diagram* selected in the General page of the Configure Control Options dialog box.

To clear the Status 0 LED:

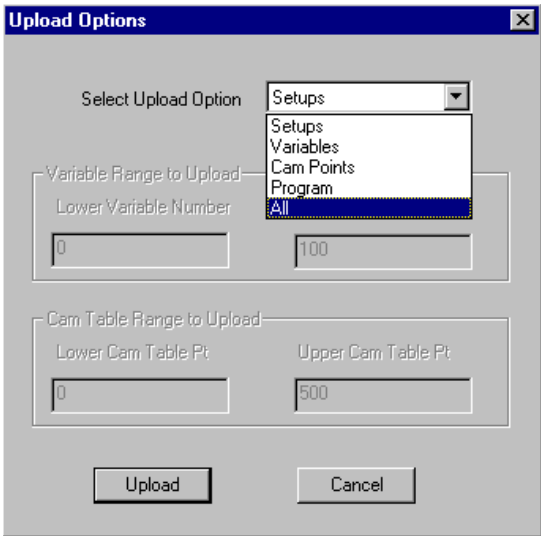
1. Wait until the parameters (described above) have been reset to the initialized values for your application.
2. Press and release the controller's RESET button.

Initializing the Software

Software initialization clears the motion controller's application and parameter memory. It is required when changing firmware revisions, and is useful any time it becomes necessary to force the motion controller into a known default state. Before software initialization, however, be sure you have saved the current setup values and the application program. Without these files, it is impossible to restore the operating configuration—or personality—of the motion controller after software initialization.

To upload the complete application program from the motion controller, including the setup values:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Upload Options**. The Upload Options dialog box appears.
3. In the *Select Upload Option* field, select **All** combo box.



4. Select **Upload**. The Upload Options dialog box closes.
5. From the menu bar, select **File**. The File menu appears.
6. Select **Save As**. The Save As dialog box appears.
7. In the *File name* field, type the name of the file.
8. Select **Save**. The Save As dialog box closes.

ATTENTION: Do not software initialize the motion controller unless you have uploaded the application program and setup data files, and saved them on your computer.

To initialize the software in the motion controller, use the iCODE .I (period followed by "I") command. Download this command to the controller in one of the following ways:

- Enter it directly using the Terminal Window.

After downloading this command, be sure to press and release the controller's RESET button to complete the download of the default parameters.

Default Data Parameters

When the iCODE .I command is executed and the controller reset, the default values for all data parameters are restored. Both the working and power-up values of the parameters for all axes are set to the values shown in the Default column of the Data Parameters table in *Appendix A* of this manual.

Default Data Bits

When the iCODE `.I` command (Initialize Control) is executed the default values are set for the Data Bits and any values previously entered for the bits are overwritten. Both the working and power-up values of the data bits for all axes are set to the values shown in the default column of the Data Bits Table in *Appendix A* of this manual.

Other Default Conditions

In addition to setting the working and power-up data parameters and data bits to the values shown in the previous tables, the `.I` command also initializes the following items.

- The position units (normally entered in the Units page of the Configure Axis Use dialog box) are set to Revs.
- The application ID string (usually entered in the Diagram Documentation dialog box) is set to App Name.
- The setup menu password (usually entered in the General page of the Configure Control Options dialog box) is reset to SET.

- The runtime display (usually set in the Configure Auto Display block) is configured to display the actual position and status of axes 0 and 1.
- The value of all user variables is set to zero.
- The value of all cam table points is set to zero.
- The application program is cleared.
- The application menu password (usually entered in the General page of the Configure Control Options dialog box) is reset to APP.
- Status 0 LED glows a continuous, uninterrupted red, except 1394GMC Turbo.

After communication is restored or the problem is fixed, the above parameters must be set back to their appropriate values for your application. You can do this manually from the motion controller's built-in setup menus, or by downloading the application program with *Download Axis/Drive Data with the Diagram* selected in the General page of the Configure Control Options dialog box.

To clear the Status 0 LED:

1. Wait until the parameters (described above) have been reset to the initialized values for your application.
2. Press and release the controller's RESET button.

Axis Locked and Axis Done Conditions

To move an axis, input new command position parameters into a selected motion block, then execute the block (or run the program). In the case of a Move Axis block, for instance, the command position is ramped up at the acceleration rate, incremented at the velocity, and decelerated at the deceleration rate specified in the block.

The motion controller compares the actual position of the axis—the current position as measured by the encoder or other feedback device—against its command position, and computes position error. The motion controller moves the axis by continually attempting to drive position error to zero.

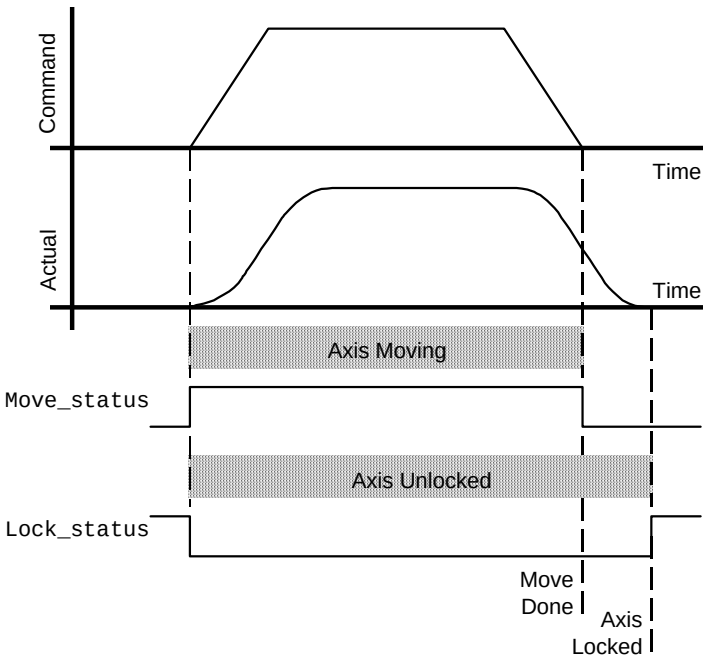
Moves, Jogs, and Time-Lock Cams

To the motion controller, a move, jog, or time-lock cam is considered done—the Move Done condition—once the command position has been reached (provided that the command position has not been changed by a subsequent Move Axis, Jog Axis, or Time Lock Cam block). At this moment, the controller stops commanding the axis to move.

However, because a small amount of position error can exist, the axis may still be moving, and its actual position changing. The axis is not considered locked (the Axis Locked condition) until both of the following occurs:

- The Move Done condition is met.
- The position error (set as Error Tolerance in the Dynamics page of the Configure Axis Use dialog box) of the axis is less than the position lock tolerance (set as Lock Tolerance in the Positioning page of the Configure Axis Use dialog box) for the axis.

For example, the figure below shows what happens when a trapezoidal Move Axis block is executed. The situation is similar for a Jog Axis or Time-Lock Cam block.



When the move is executed, the command position changes according to the parameters of the Move Axis block, resulting in the command velocity profile shown above. (Velocity profiles are used instead of position profiles because it is easier to see the form of the motion this way.) In general, the actual position lags behind the command position (the lag is exaggerated here for clarity) as a result of position error as shown in the Actual velocity profile.

While a move is in progress, Move_status = 1 (true), Axis_status = 3 (Moving), and Lock_status = 0 (Unlocked).

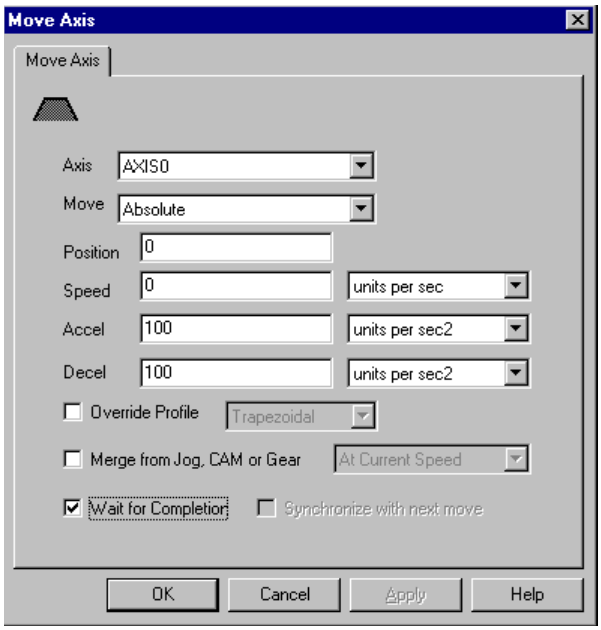
When the move is done, Move_status = 0 (false), and Axis_status = 1 (Unlocked).

When the move is done and the position error is less than the position lock tolerance, Axis_status = 0 (Locked) and Lock_status = 1 (Locked).

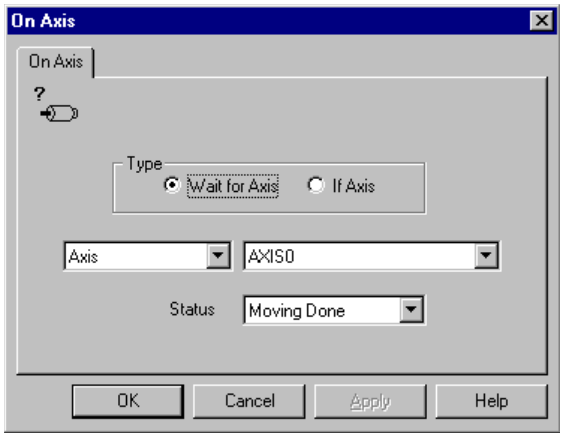
Another way to look at this is that the move done condition is the trailing edge of the Move_status variable, and the axis locked condition is the trailing edge of the Lock_status variable.

The distinction between the axis done condition and the axis locked condition is lost if a large position lock tolerance is specified. In such a case, the position error, when the move is done, is less than the position lock tolerance, with the result that the axis is locked as soon as it is done.

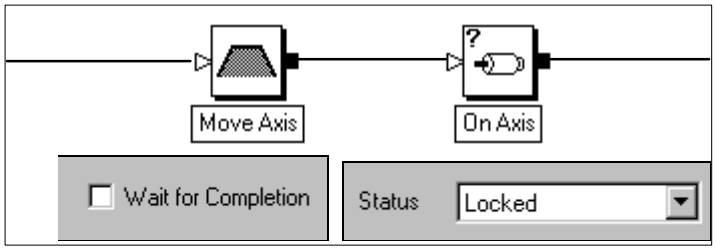
The Wait for Completion check box in the Move Axis block (shown below) waits for the move done condition—not for the axis locked condition—and no faults on the axis.



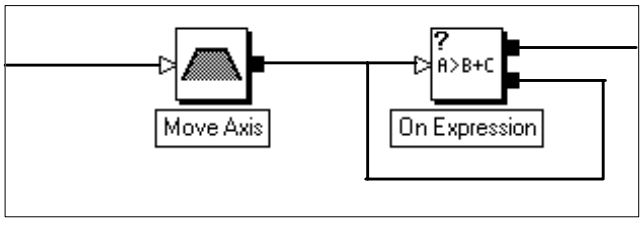
Thus, it is similar to a Wait for Axis command in the On Axis block with Moving Done (shown next) selected from the Status menu (except for the fault check).



To pause program execution until the move is done and the axis is locked, follow the Move Axis block immediately with a Wait for Axis command in the On Axis block with Axis Locked selected from the Status menu, as shown below.



For the most precise control in pausing program flow at the end of a move, jog, or time-lock cam, follow the motion block with an If Expression command in the On Expression block as shown below.



Connect the 0 (false) node of the On Expression block back to its input to form a wait for position function and enter an expression that logically combines the conditions for which you want to wait.

For example, the axis locked status condition does not check whether the velocity of the axis is zero. If an axis has significant overshoot or ringing at the end of the move, the axis locked condition becomes true before the actual motion has settled out. In this case, use an expression of the following form in the On Expression block to wait for the velocity of the axis to be zero and the axis to be locked.

(Average_Velocity_AXIS0 = 0) && Lock_status_AXIS0

This technique can be extended to wait for other combinations of conditions for many different applications.

Moves, Jogs, and Time-Lock Cams with Gearing, Position-Lock Cams, or Interpolated Motion

When a move, jog, or time-lock cam is used in conjunction with gearing, or a position-lock cam, or interpolated motion on the same axis, the situation is a little different. In this case, the axis is locked (Lock_status = 1) whenever the position error is less than the position lock tolerance, regardless of whether the move, jog, or time-lock cam is done or not.

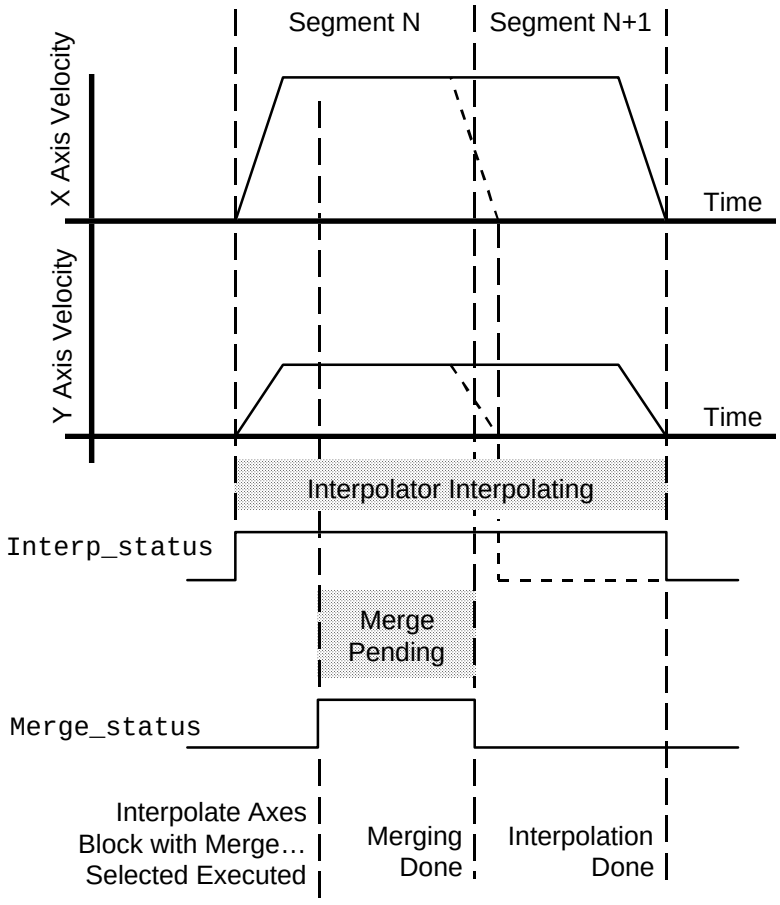
Why the difference? Because in incremental motion applications (where moves, jogs, and time-lock cams are used without gearing or position-lock cams to position the axis to some specific point), it is most important to know when the motion has finished and the axis is in position. The fact that it is in position during the motion is of no consequence.

Conversely, in continuous process applications where gearing or position-lock cams are active, moves, jogs, and time-lock cams are used to adjust synchronization between moving axes. In these applications it is most important to know whether the axis is in position (locked) at all times regardless of what motions are being commanded. Thus the Lock_status variable and the related axis locked conditions are designed to function differently depending on how the axis is being used.

Interpolated Motion

Like single-axis moves, jogs, and time-lock cams, interpolated motion is considered done when the command position of any of the specified axes is no longer changing as a result of an Interpolate Axes block. In other words, the interpolator has finished commanding the axes to move. When two interpolated motion segments are merged together, the interpolated motion is not done until the final merged segment is done.

For example, the following figure shows what happens when a two-axis linear move is merged into another two-axis linear move at the same speed.



The dotted line in the motion profiles shows where Segment N would have ended if it had not been merged into Segment N+1 and the corresponding Interpolation Done point.

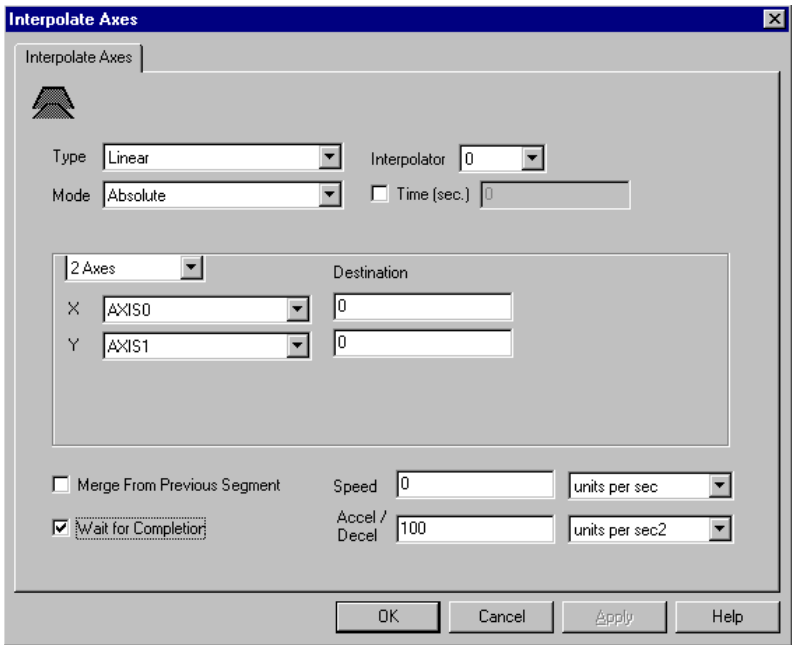
While an interpolated move is in progress, Interp0_status or Interp1_status = 1 (true) depending on the specified interpolator and Axis_status ≤ 5 for each specified axis if no faults are active on the axis. The status of each of the specified axes is normally either locked (Lock_status = 1 and Axis_status = 0) if the position error is less than the position lock tolerance or unlocked (Lock_status = 0 and Axis_status = 1) if the position error is greater than the position lock tolerance.

When the interpolated move is done, Interp0_status or Interp1_status = 0 (false) depending on the specified interpolator. Another way to look at this is that the interpolation done condition is the trailing edge of the appropriate Interp_status variable, as shown in the previous figure.

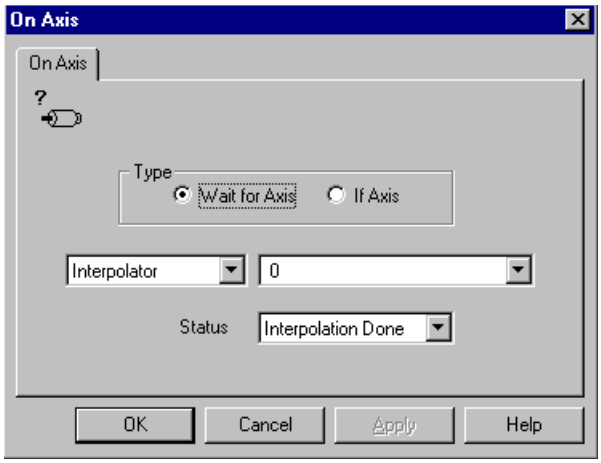
After an Interpolate Axes block with Merge From Previous Segment selected is successfully executed, Merge_status_Interp0 or Merge_status_Interp1 = 1 (depending on the specified interpolator) and the merged motion is pending.

When the currently executing interpolated motion is done, Merge_status_Interp0 or Merge_status_Interp1 = 0. Another way to look at this is that the merging done condition is the trailing edge of the appropriate Merge_status variable, as shown in the previous figure.

The Wait for Completion check box in the Interpolate Axes block (shown below) waits for the interpolation done condition and no faults on any of the specified axes.



Thus, it is similar to a Wait for Axis command in the On Axis block with the Status Interpolation Done selected from the Status menu except for the fault check (shown below).



The technique discussed earlier, for waiting until the motion is done and the axes are locked or are at zero speed, can also be used with interpolated motion.

Merging Different Motion Types

Merge From Jog, CAM, or Gear

When merging a slave axis from actual-position gearing, or camming to a move or jog by selecting Merge from Jog, CAM or Gear and At Current Speed, in a Move Axis or Jog Axis block in GML Commander, as shown below, the speed of the move or jog is not always exactly the same as the previous gearing speed.

In fact, if the master speed is low enough, occasionally the move or jog doesn't work at all—the axis just sits there and doesn't move. The reason is that it has to do with how axis velocity is calculated inside the motion controller.

The actual velocity of an axis (used when merging at current speed) is the change rate of axis position every servo update. Since position can only change by whole counts, the resolution of the actual velocity value is ±1 count per servo update. For example, with a servo update rate of 1 kHz and 4,000 counts per revolution, actual velocity resolution is shown in the following equation:

$$\pm 1 \frac{\text{Count}}{\text{Update}} \text{ x } \frac{1 \text{ Revolution}}{4000 \text{ Counts}} \text{ x } \frac{1 \text{ Update}}{0.001 \text{ Second}} \text{ x } 60 \frac{\text{Seconds}}{\text{Minute}} = \pm 15 \text{ RPM}$$

If the speed of the axis lies somewhere between multiples of 15 RPM, the measured actual velocity varies from update to update such that it averages out to the correct speed. For example, 2,000 RPM is between 1,995 (15 x 133) and 2,010 (15 x 134) RPM. So, with the axis moving at 2,000 RPM (due to a Gear Axes or Position Lock Cam block), the motion controller calculates velocities of 1,995 RPM, 1,995 RPM, 2,010 RPM, 1,995 RPM, 1,995 RPM, and 2,010 RPM for successive updates. Taken together it averages 2,000 RPM.

However, when the move or jog takes over from gearing or camming, the velocity used is either 1,995 or 2,010 RPM (whichever is the most recent sample), not exactly 2,000 RPM as expected. If the master's speed is less than 15 RPM, sometimes the measured actual velocity is zero. If the most recent sample, when the merged move or jog is initiated, has measured zero velocity, the axis does not move as expected. Actually, as far as the motion controller is concerned, it is moving at zero speed.

There are two solutions to this problem:

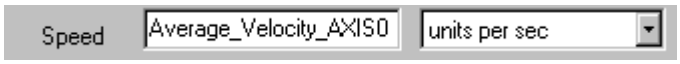
- Gear or cam to command position instead of actual position.
- Merge to a move or jog at a programmed speed equal to the average_velocity variable for the axis.

When gearing or camming to command position, use the command velocity. This is more precise than the actual velocity, because it is generated internally as a floating point number (rather than the accumulation of individual encoder counts). To gear to command position, select Command Position from the Slave to menu in the Gear Axes function block.

The Average_Velocity for an axis is calculated by averaging the measured actual velocities over the averaged velocity timebase (specified in the motion controller's application setup menu). Therefore, its resolution is ±1 count per averaged velocity timebase, rather than ±1 count per servo update. Continuing our example, an averaged velocity timebase of 0.1 seconds results in an average velocity resolution of the following, that is better by a factor of 100.

$$\frac{\text{Count}}{0.1\ \text{Seconds}} \times \frac{1\ \text{Revolution}}{4000\ \text{Counts}} \times 60\frac{\text{Seconds}}{\text{Minute}} = \pm 0.15\ \text{RP}$$

Thus, at 2,000 RPM, merging from gearing to a move at a programmed speed of Average_Velocity for the slave axis results in a perfectly smooth transition. For example, if Axis 0 is the gearing slave, the Move Axis block's Speed data entry input box would look like this:



The Merge from Jog, CAM or Gear check box and menu selection would look like this:

.

Going Online

You can build and edit your diagram, create variables, and test program syntax without being online with your controller, but you must go online to test the logic of your program. Also, certain Axis Use options are only available when you are online.

This chapter contains step-by-step procedures for going online. The topics in this chapter are:

- Using the online toolbar
- Accessing your controller
- Translating a diagram to a program and downloading
- Choosing the information view mode
- Running a program
- Monitoring program flow
- Monitoring variable, I/O, and general watch status
- Send commands directly to the motion controller
- Upload options

Using the Online Toolbar

The online toolbar buttons, like their menu command counterparts, enable you to communicate with the motion controller. You use these commands to translate diagrams to programs that are downloaded and executed.

The online toolbar provides the same functionality of the Diagram menu options with the exception of the *Select Direct Command*, which you cannot select from a menu. As the name suggests, you use this field to select and open a direct command dialog box.

Online button	Diagram menu command	Description
	Kill Control	Kills any motion and stops program execution. The CPU relay is opened and the system OK light is turned off. To recover from this condition, press the reset button on the controller.
	Pause	Toggles the program execution to suspension or restart
	Stop	Aborts program execution
	Go	Starts program execution
	Single Step	Executes one block of the program
	Auto Step	Shows program flow at slower than normal program speed
	Trace	Periodically shows program flow while running at full program speed
	Insert/Remove Breakpoint	Inserts or removes a breakpoint from the currently selected block

Online button	Diagram menu command	Description
	Clear All Breakpoints	Clears all breakpoints from the currently selected diagram
	Online Connection	Toggles the communication with the motion controller on or off
	Diagram Download	Translates and downloads the diagram to the motion controller
	Upload Options	Allows upload of program scripts, variables, cam points, or all
	Debug Mode	Toggles the debug mode to normal (English text, diagram trace) or terminal (iCODE text, trace)
	Select direct command	Select a direct command from list of online commands to execute
	Execute Direct Command	Opens the dialog box for the direct command in the list of direct commands
	Re-execute Direct Command	Executes the last executed direct command again

Accessing Your Controller

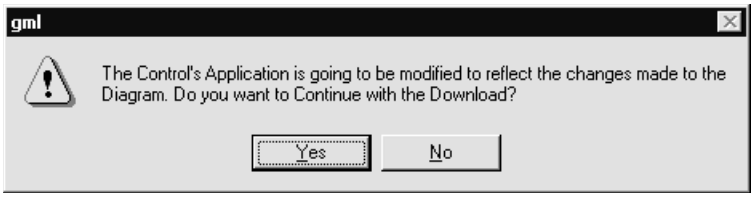
You can access your controller either by pressing the Online Connection button on the Online Toolbar or by selecting **Online Connection** from the Diagram menu. First you must turn on the Controller. When communication with your motion controller is enabled, the command/response links with the controller appear in the Terminal Window.

Translating a Diagram to a Program and Downloading

A diagram must be downloaded to the controller before it can be used. The download process translates the diagram to program script and downloads the script to the controller. If you make functional changes to your diagram, you must download it again before you run it. You do not need to download if the changes are nonfunctional changes, such as dragging blocks or connections.

To download the diagram appearing in the workspace:

1. Select **Diagram Download** from the **Diagram** menu.
2. The download process begins. A dialog box appears requesting confirmation to download.



3. Select **Yes** to continue with the download.
4. Once the download has completed successfully, you can do one of the following:

- Select Go to run the program.
- Select Single Step, Auto Step, or Trace to monitor the program.
- Set breakpoints in the program.

Note: If you want to change your program's axis/drive setup data, this information is not automatically downloaded to the controller unless Download Axis/Drive Data with the Diagram is selected in the Configure Control Options dialog box. See the *Downloading Axis and Drive Setup Data* section.

Downloading Axis and Drive Setup Data

To directly override the axis/drive setup data resident in the controller, select the configuration option that controls this function in the Configure Axis Use dialog box.

To download axis/drive data directly:

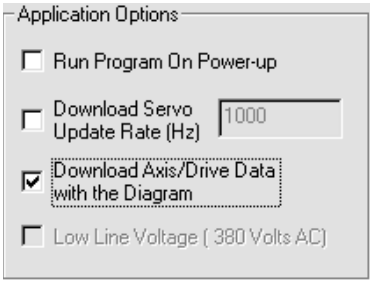
1. Select **Axis Use** from the **Configure** menu.
2. The Configure Axis Use menu appears.
3. Select the axis. The dialog box appears.
4. In the *Axis Use Configuration* area on the Apply page, select **Download**.

ATTENTION: Before you select the Axis/Drive Data Download option, be sure you have valid axis/drive data in the Axis Use dialog box or unexpected motion can occur.

To download the saved axis/drive data with your diagram:

1. Select **Control Options** from the **Configure** menu. The Configure Control Options dialog box appears.

2. Select **Download Axis/Drive Data in the Diagram** in the *Application Options* area. A checkmark appears in the box.



3. Select **OK**.

Running a Program

GML Commander provides you with several ways that you can start and stop a program that has been downloaded to the controller:

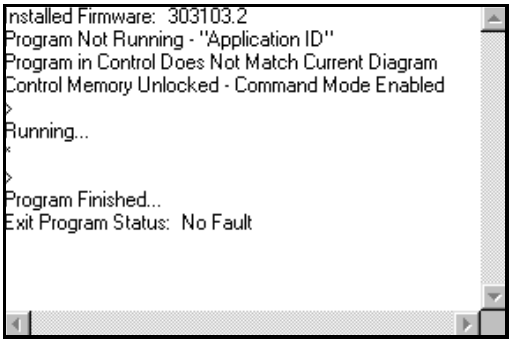
- Start a program
- Pause a program
- Resume a paused program
- Stop a program
- Stop a program and kill motion
- Run program on power-up

Starting a Program

To start a program:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Go**. The program runs.

The Terminal Window below shows an example of a program that ran successfully:



Important: If you select Go while in Trace mode, Trace mode is aborted but the program continues.

Pausing a Program

To pause an executing program, select **Pause** from the Diagram menu. The program pauses.

Important: Continue program execution by selecting the Resume, Go, or Trace command.

Resume the Execution of a Paused Program

To resume execution of a paused program, select **Resume** from the Diagram menu. The program continues execution at the point where it paused.

Important: The program restarts from the point of suspension in the current Debug Mode. This is useful when a program is paused as a result of a breakpoint. See Setting a Breakpoint for more information.

Stopping a Program

To stop a program, select **Stop Program** from the Diagram menu. The program stops, motion stops and program execution is aborted.

Important: To begin again, the program must be restarted from the beginning.

Stop a Program and Kill Motion

The Kill Control command performs several functions.

To stop the program, open the enabled watchdog and drive enable relays, disable feedback, and zero the commanded servo output for all axes:

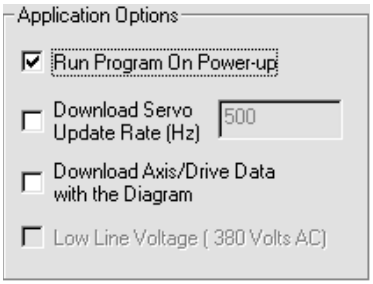
1. From the menu bar, select Diagram. The Diagram menu appears.
2. Select **Kill Control**. Program execution is aborted, the motion controller opens the watchdog relay and disables all drive enable relay, disables feedback, and zeroes the commanded servo output for all axes.

Note: The only way to recover from a Kill Control action is to cycle power or reset the controller.

Run Program on Power-up

To cause a program to run on power-up:

1. From the menu bar, select **Configure**. The Configure menu appears.
2. Select **Control Options**. The Configure Control Options dialog box appears.
3. Select **Run Program On Power-up** in the Application Options area. A check mark appears in the box.



4. Select **OK**.
5. Download the diagram.

Monitoring Program Flow

GML Commander provides several options for monitoring the program flow of a program that has been downloaded to the controller:

- Trace through a program at full speed
- Trace though a program one block at a time
- Trace though a program slowly
- Set a specific breakpoint to show program command execution
- Clear all breakpoints in the program

To show program flow of the downloaded program use the Debug Mode command on the Diagram menu, to toggle from Normal Mode view to Terminal Mode view.

- Select Normal Mode view to display program flow in the Diagram Window.
- Select Terminal Mode view to display program flow in the Terminal Window.

Select How to View Information

You use the Debug Mode command to toggle between normal or terminal mode view. When stepping, auto stepping, or tracing, the flow of the program execution appears graphically in the Diagram Window or textually in the Terminal Window. You cannot view program flow in both windows at the same time.

To select normal or terminal mode:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Debug Mode**. The view mode changes.

If the view mode is:	It becomes:
Normal	Terminal. This means that iCODE appears in the Terminal Window. Use this view for textual tracing of program execution in the Terminal Window.
Terminal	Normal. This means that English text appears in the Terminal Window. Use this view for full graphical tracing of program execution in the Diagram Window.

Using Trace

To show program flow of the downloaded program appearing in the workspace, select **Trace** from the Diagram menu.

Important: Because trace runs at full speed, select Single Step or Auto Step to monitor the program at a slower pace.

Using the Normal Debug Mode to Monitor the Program Blocks Being Traced

To trace the execution of a program:

1. If your diagram has not been downloaded, from the Diagram menu, select **Download Diagram**. The program is sent to the controller.
2. Select **Trace**. The program runs at full speed and periodically highlights blocks in the Diagram Window as they are executed.

Important: In Normal mode, the Diagram in the workspace must match the program resident in the controller.

Terminating Trace Mode

To terminate trace mode:	Select:
Without aborting the program	**Go** while it is tracing.
And abort program execution	**Stop**.

Using Auto Step

Auto Step shows program flow at slower than normal program speed. When execution of each block is complete, the next block in the program flow is highlighted. This continues until the program terminates.

To automatically step through each program block:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Auto Step**. Commands appear in the Terminal Window.

Using Single Step

Single Step shows program flow at the slowest speed. When execution of a block is complete, the next block in the program flow is highlighted. At this point, to continue, you must select Single Step again.

To step through program blocks one at a time:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Single Step**. Commands appear in the Terminal Window.

Setting a Breakpoint

Use the Breakpoint option to set or clear breakpoints. You can set and clear breakpoints from the Diagram menu only after the diagram is downloaded. GML Commander supports up to 10 breakpoints.

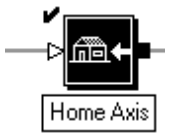
When a block has a breakpoint set, its validation checkmark is highlighted in a black square. When a breakpoint is reached, the block with the breakpoint is highlighted.

Note: Every time you make a functional change to a diagram, you need to download the diagram again.

Important: It is possible to continue execution through breakpoints (trap mode) or to stop at a breakpoint. See *Resuming the Execution of a Paused Program* for information about resuming a program.

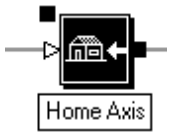
To set a breakpoint in a downloaded diagram:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Online Connection**.
3. From the menu bar, select **Diagram**. The Diagram menu appears.
4. Select **Download Diagram**.
5. Select the block where you want to set a breakpoint. The block and the green check at the upper-left corner of the block appear highlighted.

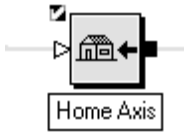


6. From the menu bar, select **Diagram**. The Diagram menu appears.

7. Select **Insert/Remove Breakpoint**. A black validation square appears over the checkmark. The breakpoint is set.



8. Click the left mouse button somewhere in the open area of the Diagram Window. The normal color of the block reappears and the check mark is shown against the backdrop of a black square.



Clear Breakpoints

To clear breakpoints:

1. From the menu bar, select **Diagram**. The Diagram menu appears.
2. Select **Clear All Breakpoints**. The breakpoints are cleared.

Monitoring Variable, I/O, and General Watch Status

You can monitor system variables or selected variables and I/O values from the General Watch list in the Tag Window. For information on adding them to or deleting them from the General Watch list, see *Defining Variables, Constants, and I/O*.

Important: You must be online with the controller to monitor.

Sending Commands Directly to the Motion Controller

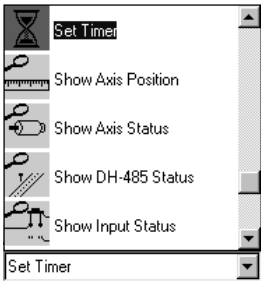
The Direct Command function allows you to issue commands directly to the motion controller, without having to redesign your diagram and download it to the controller.

Important: You can only select the *Select Direct Command* field from the online toolbar. Refer to the *Using the Online Toolbar* section for more information.

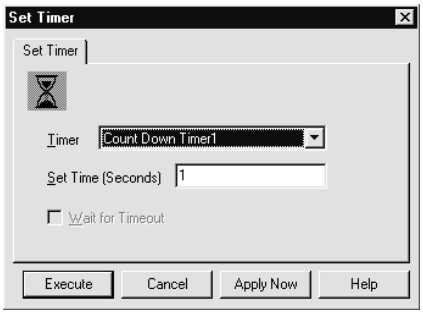
To communicate directly with your motion controller, use the Direct Command function located on the online toolbar:

To select a command to send directly to the controller:

1. From the Direct Command menu located in the online toolbar, select the display values arrow. A list of the GML Commander direct commands appears.



2. Select a direct command. The direct command's dialog box appears.



Note: For information about each direct command, see *Function Blocks* in the *GML Commander Reference Manual* (publication GMLC-5.2).

3. Configure the dialog box with appropriate information.

4.

If you select:	The following occurs:
Execute	The block's dialog box closes. The *Reexecute Direct Cmd* command on the Diagram menu and its corresponding button on the online toolbar are reenabled. In the Terminal Window, the command sent to the controller appears: • In English if Normal Mode is selected. • In iCODE if Terminal Mode is selected.
Apply Now	The block's dialog box remains open to allow you to: • Reselect the Apply Now command to execute the block again. • Change the fields of the block's dialog box before reselecting the Apply Now command. In the Terminal Window, the command sent to the controller appears: • In English if Normal Mode is selected. • In iCODE if Terminal Mode is selected.

5. When the direct command's dialog box closes, you can do one of the following:

   - From the Diagram menu, select **Execute** to redisplay the direct command's dialog box to change your previous settings and execute the function direct command again.
   - From the Diagram menu, select **Reexecute** to execute the last direct command again.

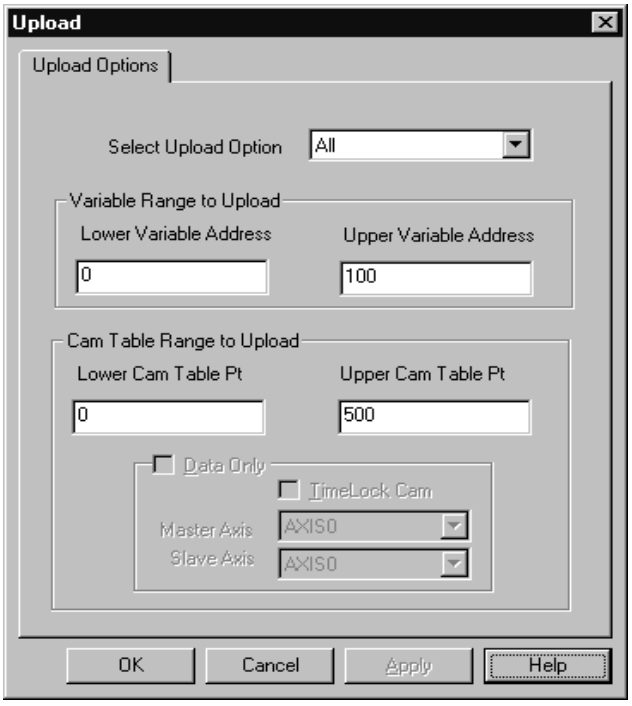
Upload Options

Use the Upload Options function to transfer a script file from the controller to GML Commander. This file can then be stored for later use. Stored files can be retrieved and downloaded to another controller using GML Commander

To transfer a script file from the controller to GML Commander:

1. From the menu bar, select **Diagram**. The Diagram menu appears.

2. Select **Upload Options**. An Upload Options dialog box similar to the following appears:



3. Make an entry in the following field:

Field	Description	
Select Upload Option	Select one of the following:	
	Setups	Upload the controller setup portion of the script only.
	Variables	Upload the user variables of the script only.

Field	Description	
	Cam Points	Upload the cam points of the script only.
	Program	Upload the executable portion of the program only.
	All	Upload the script.

4. Make entries in the following fields in the *Variable Range to Upload* area:

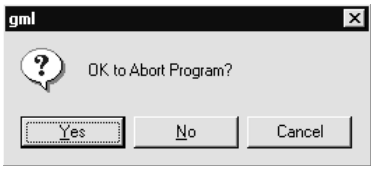
Field	Description
Lower Variable Address	Type the address of the lowest variable that you want to upload.
Upper Variable Address	Type the address of the highest variable that you want to upload.

5. Make entries in the following fields in the *Cam Table Range to Upload* area:

Field	Description
Lower Variable Number	Type the address of the lowest Cam table variable that you want to upload.
Upper Variable Number	Type the address of the highest Cam table variable that you want to upload.

6. Select **Upload**.

Important: Uploading can be done only if the program is not executing. If your program is currently running in the controller and you want to abort the program upload, an inquiry box allows you to confirm this action.



If you select yes, the program currently running in the controller stops to allow the uploading process to occur.

The program in the controller appears.

See the *On-Line Help* for more detailed information about Uploading Options.

CPU Utilization

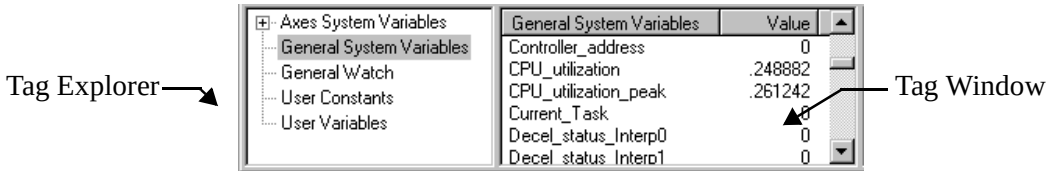
Two of the most valuable features of the controllers are the CPU_utilization variable, and the CPU_utilization_peak variable.

The CPU_utilization variable shows how much of the motion controller's CPU (Central Processing Unit, or microprocessor) capacity is being used to generate and control motion, and thus how much is left for program execution and external communication (RIO, SLC, DH-485, AxisLink I/O Read Remote, and Serial Ports.). If CPU Utilization gets too high, program execution and communication slow down dramatically, adversely affecting application performance.

The CPU_utilization_peak variable (enabled only in iCODE versions 3.5 and higher) displays the historic high value for the CPU_utilization value. This value is reset at power up, and when you press the motion controller's Reset.

To monitor the current CPU Utilization and CPU Utilization Peak variables:

1. Be sure you are operating Online by selecting the **Online Connect** button in the Online Toolbar, (or by selecting **Online Connection** in the Diagram Menu, or by pressing **F8**).
2. Select **General System Variables** in the Tag Explorer.
3. In the Tag Window, scroll down to the CPU_utilization and CPU_utilization_peak variables, as shown below.

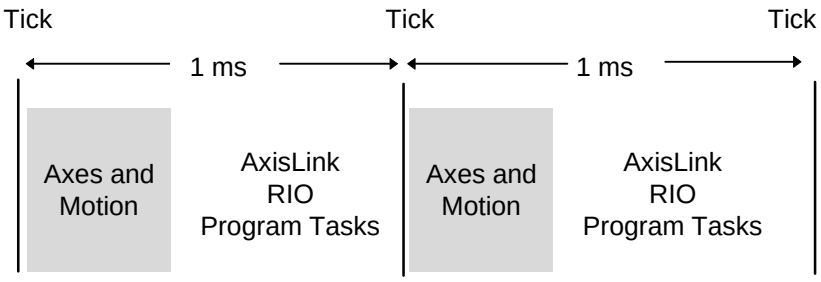


These variable values are shown as a normalized percent (between 0 and 1). For example, the CPU Utilization value shown in the example, above, indicates that nearly 25% of the CPU capacity is currently being used (thus, 75% of CPU capacity is available), and peak CPU utilization has been only about 26%.

Understanding CPU Utilization

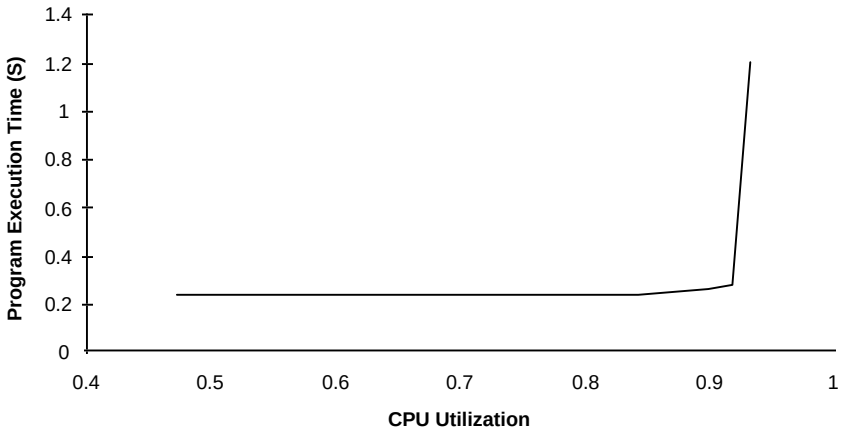
To understand the effect of the CPU Utilization on program execution and communication speed, you need to know how the motion controller works. Internally, any motion controller lives by a clock, whose rate is determined by the setting of the Servo Update Rate (parameter D47) in the General page of the Configure Control Options dialog box, or with a Control Settings block.

For example, if the Servo Update Rate is set to 1,000 Hz, the time between ticks on the clock is 1 ms, as shown below.



At every tick of the clock, the motion controller stops whatever it is doing and performs all the calculations necessary to keep the axes moving as commanded by the application program. Thus, as more and more axes and motion (moves, jogs, cams, etc.) are added to the application, less and less time is available for external communications and program execution.

Normally this is not a problem, but as CPU utilization exceeds 80 – 90% of capacity, program execution can grind to a virtual halt and external communications can fail. Until this point is reached, however, adding axes and motion has very little effect upon program execution and external communications, as shown in the following graph.



This graph was generated by timing a small application program running in the motion controller (using the Free Running Timer) as various additional axes and motion were added. As you can see, there is virtually no effect on program execution speed until a certain threshold is reached—and then the effect is dramatic!

The threshold where program execution grinds to a halt depends mostly on whether you are using RIO and/or AxisLink. Approximate program execution thresholds are given in the table below.

CPU Utilization Program Execution Thresholds

Using RIO	Using AxisLink	CPU Utilization Threshold
No	No	90%
No	Yes	80%
Yes	No	90%
Yes	Yes	70%

Effects of Excessive CPU Utilization

If CPU Utilization in your application exceeds the thresholds set forth above, program execution speed slows drastically, and external communications can fail.

Also, when the CPU_utilization variable value reaches 90%, the motion controller (using firmware version 3.5 or higher) executes a Kill System command, and generate a CPU Utilization Overrun fault (Global Fault 15), which in turn triggers the following events:

- The CPU Watchdog is disabled, with the result that the green System OK LED turns off (or turns red, depending on the version of your controller).
- Feedback for all axes is disabled.
- Motion on all axes is disabled.
- All servo outputs are set to zero.
- The drive enable outputs are disconnected.

As a result, CPU utilization is lowered to an acceptable rate, and serial port communication is restored. The CPU_utilization_peak variable retains and displays the highest value reached by the CPU_utilization variable.

See *Connecting the CPU Watchdog* in the Installation and Hookup section of your controller's *Installation and Setup Manual*.

To clear the CPU_utilization_peak variable, execute an Equation function block using the Online Toolbar:

1. Select **Type: Configured**.
2. Select **General System Variables**.
3. Set the CPU_utilization_peak variable equal to zero.

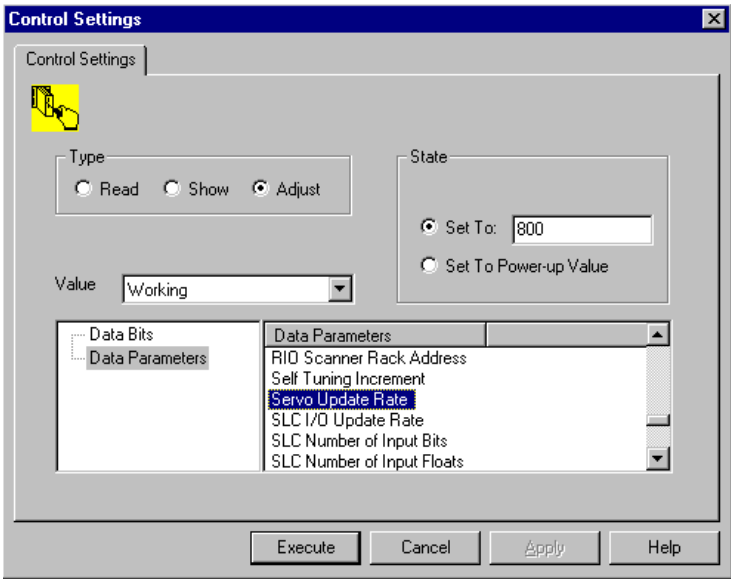
Decreasing the Servo Update Rate

To reduce the CPU utilization rate, and avoid the effects of excess CPU utilization described in the previous section, do one of the following:

- Decrease the servo update rate.
- Eliminate features that directly add to CPU utilization.

To decrease the servo update rate, execute a Control Settings block in the Online Toolbar, as follows:

1. In the Online Toolbar's Select Direct Command window, select **Control Setting**. The Control Settings dialog box appears.



2. Make the following field entries:

Field	Description
Type	Select **Adjust**.

Field	Description
Value	Select **Working**.
Tag Explorer	Select **Data Parameters**.
Tag Window	Select **Servo Update Rate**.
State	• Select **Set To**. • Type the rate in Hz.

3. Select **OK**.

Example

The Control Setting block parameters, shown above, set the servo update rate to 800 Hz, resulting in 1.25 ms between clock ticks.

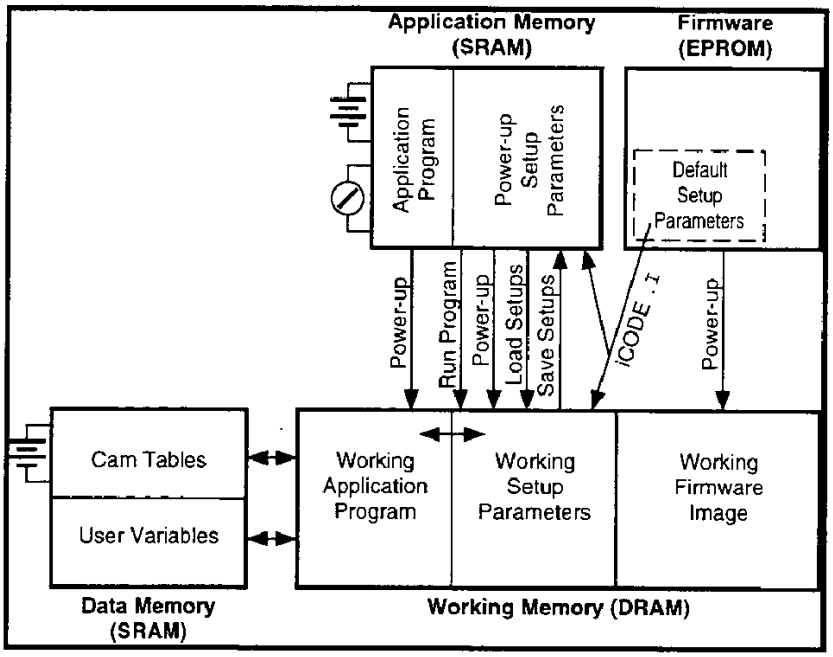
The change takes effect immediately, and you can then examine the new CPU Utilization value as explained earlier. The Servo Update Rate is not axis-specific; the same rate is used by all axes.

A good rule of thumb is to reduce the servo update rate in increments of 100 Hz until the CPU utilization is less than the threshold values given previously. When you have settled on the required rate, adjust the data parameter by executing one last Control Settings block, selecting Set to Power-up Value, thereby making the change permanent.

In addition, once the optimal servo update rate has been determined, this value should be set in the General page of the Configure Control Options dialog box. This way, if the controller ever needs to be initialized or exchanged, the servo update rate is set correctly in the new controller.

Memory Organization

There are four separate memory areas in the motion controller, each with a different function. These four memory areas are shown in the following diagram.



Firmware

The Firmware, stored in EPROMs, is downloaded to working memory (dynamic RAM) whenever you apply power to the controller, or press the RESET button. This allows the firmware to execute much faster than if the EPROMs were accessed directly.

Note: You can only change firmware by replacing the EPROMs or, in the 1394 GMC Turbo, by performing a Flash EPROM update procedure.

Application Memory

The Application Memory is battery-backed (as shown by the battery symbol ☰) static RAM, that can be locked (as shown by the lock symbol ⊘) via the keyswitch on the 1394 GMC, Compact (IMC-S/23x), or Integrated (IMC-S/21x) units, or by removing the memory unlock jumper on the Basic (IMC-S/20x) unit. This non-volatile memory stores the application program and the power-up setup parameter values. When you apply power to the controller, or when you press the RESET button, both the application program and the setup parameters values stored in application memory are downloaded to working memory. When you run the program, the setup parameter values stored in application memory are again downloaded to working memory and used. Likewise, it is the working setup parameter values that are used when feedback is enabled for each axis.

Data Memory

Data Memory, also battery-backed (but not write-lockable) static RAM, is used to store the cam tables and user variable values. In the 1394 GMC/ GMC Turbo and Compact (IMC-S/23x), this memory also stores the local DH-485 data files. These values are read and written by the application program as it executes.

Working Memory

Working Memory is the volatile dynamic RAM memory actually used by the motion controller’s microprocessor. It stores an image of the firmware, an image of the stored application program, and the working setup parameter values.

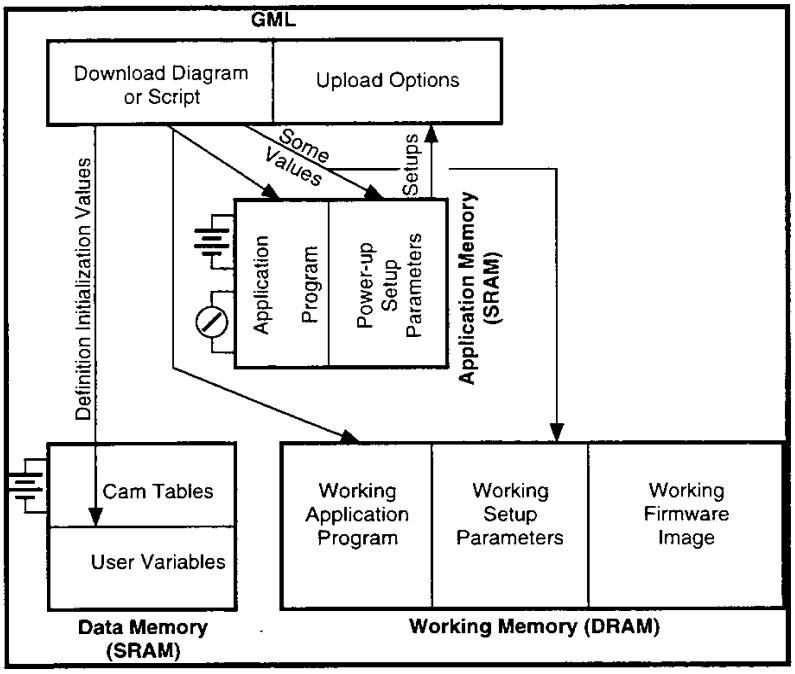
How the Motion Controller Uses Memory

When the setup menus in the motion controller are used to edit the setup parameters, the question "Load Setups from App Module?" is asked. Answering "YES" copies the power-up setup parameter values from the application memory into working memory. While in the setup menu, it is the working parameter values that change, not the power-up values. At the conclusion of the setup menus, the question "Save Setups to App Module?" appears. Answering "YES" copies the working setup parameter values (presumably now properly set) into application memory for use whenever the controller powers-up. Because working memory clears whenever either power to the motion controller ceases or you press the RESET button, always save the working setups into application memory after using the setup menus.

A default set of values is contained in the firmware for initializing the setup parameters to a known state whenever necessary. The iCODE .I command stores these default setup parameter values in both the application and working memory. This is why you need to unlock the memory before using the **.**I command. See the *Initializing Your Motion Controller* chapter for more information on the **.**I command.

How Commander Uses the Motion Controller's Memory

Commander also accesses these three memory areas when diagrams, scripts, etc. are uploaded and downloaded, as shown below.



When you download a Commander diagram or script to a motion controller, the application program (generated from the Commander diagram) is stored in both the application and the working memory, erasing whatever program was previously stored there. Also, the initial value of all user variables, initialized in the User Variables definition, is stored directly in data memory. Finally, certain setup parameters are stored directly in application and working memory. The setup parameter values downloaded by Commander overwrite the values set in the application menu.

The Online Toolbar's Upload Options button (along with the Diagram Menu's Upload Options selection, and Ctrl + F9) let you upload

- The setup values from application memory (the power-up values).
- The cam tables and user variables directly from data memory.
- The application program from application memory.

Appendix **A**

Data Parameters and Data Bits

Data Parameters are variables that are used to configure the controller options. Data Bits are Boolean values (0,1) which further define the configuration values for the controller by disabling or enabling controller options.

Displaying Data Parameters

To display Data Parameters in the Tag Window:

1. In the Tag Explorer, under Data Parameters, select either an Axis or General Parameters (i.e., non-axis-specific data parameters). The data parameters associated with that selection appear in the Tag Window.

Note: If an axis does not appear, it has not been enabled in the Configure Control Options dialog box.

2. In the Tag Window, scroll down the list until the desired data parameter appears.

Displaying Data Bits

To display Data Bits in the Tag Window:

1. In the Tag Explorer, under Data Bits, select either an Axis or General Bits (i.e., non-axis-specific data bits). The data bits associated with that selection appear in the Tag Window.

Note: If an axis does not appear, it has not been enabled in the Configure Control Options dialog box.

2. In the Tag Window, scroll down the list until the desired data bit appears.

The following tables are listed by the data parameter or data bit description displayed in Commander when a Control options Block is selected. They are arranged alphabetically. Some data parameters and bits are dependent on whether Read, Show, or Adjust is selected in the Type field of the Control Options screen.

ATTENTION: We recommend using extreme caution when configuring the Adjust Control Settings type block, because carelessly changing certain Data Bits or Data Parameters can cause uncontrolled motion.

ATTENTION: Do not enter adjusting values for data parameters that do not fit the selected parameter's Value Format. See the Reference Manual for each parameter's Value Format.

Data Parameters Table

In the following table, the Value Format column displays the required numerical format for the value. A Value Format of the form X.Y means that up to X total digits, with Y digits to the right of the decimal point, may be used to specify the value. Do not enter values which do not fit the selected format or unpredictable operation may occur. Parameters marked with an asterisk (*) are not axis-specific. The Default column of this table displays the value that is set when the iCODE .I command is executed and the controller reset.

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Accel. Units Decimal Digits	D51	$0 \leq D51 \leq D50$	2.0	0
Accel. Units Total Digits (Field Length)	D50	$1 \leq D50 \leq 15$	2.0	4
Analog Test 0 Axis 1394	D101	0 = Axis 0 1 = Axis 1 2 = Axis 2 3 = Axis 3	1.0	0
Analog Test 0 Mode 1394	D99	0 = Generic 1 = Velocity 2 = Torque	1.0	1 = velocity
Analog Test 1 Axis 1394	D102	0 = Axis 0 1 = Axis 1 2 = Axis 2 3 = Axis 3	1.0	0
Analog Test 1 Mode 1394	D100	0 = Generic 1 = Velocity 2 = Torque	1.0	2 = Torque
Average Velocity Timebase	D35	Seconds	5.4	0.5 (Seconds)
Axis Configuration	D31	0 = Not Used 1 = Master Only 2 = Servo	1.0	2 (Servo) 0 (Not Used)
AxisLink Controller Address	D61	$0 \leq D61 \leq 15$	2.0	0
AxisLink Node Number	D129	$0 < D129 < 16$	2.0	0
AxisLink Nodes*	D63	$0 \leq D63 \leq 65024$	5.0	0 (No AxisLink I/O used)
AxisLink Physical Axis	D62	$0 \leq D62 \leq 3$	1.0	0 (Axis 0)

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Backlash Compensation Mode	D46	0 = Disabled 1 = Unidir. App. 2 = Load Rev.	1.0	0 (Disabled)
Backlash Offset	D14	Position Units	D29.D30	0
Channel A Baud Rate*	D32	0 = 300 Baud 1 = 1200 Baud 2 = 2400 Baud 3 = 4800 Baud 4 = 9600 Baud 5 = 19,200 Baud 6 = 38,400 Baud 7 = 56k Baud 8 = 115.3k Baud 9 = 128k Baud	1.0	4 (9600 Baud)
Channel B Baud rate*	D33	0 = 300 Baud 1 = 1200 Baud 2 = 2400 Baud 3 = 4800 Baud 4 = 9600 Baud 5 = 19,200 Baud 6 = 38,400 Baud 7 = 56k Baud 8 = 115.3k Baud 9 = 128k Baud	1.0	4 (9600 Baud)
CNET MAC ID	D144	Integer from 1 to 99	2.0	1
CNET Number of Input Bits	D140	8, 24, or 40	2.0	8
CNET Number of Input Floats	D141	Integer from 0 to 15	2.0	1
CNET Number of Output Bits	D142	8, 24, or 40	2.0	8
CNET Number of Output Floats	D143	Integer from 0 to 15	2.0	1

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Converter Card	D114	0 = AEC 1 = TEC 2 = REC 3 = REC - 4096@10 4 = REC - 4096@12 5 = REC - 4096@14 6 = REC - 4096@16	1.0	4 (0 for the 1394)
Current Negative Limit 1394	D75	% IRated	3.n	300
Current Positive Limit 1394	D74	% IRated	3.n	300
Current Preload 1394	D77	% IRated	3.n	0
Current Rate Limit 1394	D76	% IRated / mSec	4.n	100
DB Compensation (±)	D4	Volts	5.4	0 (Volts)
Default Homing Mode	D36	0 = Passive 1 = Absolute 2 = Active 3 = Absolute MV 4 = Absolute serial	1.0	2 (Active) 0 (Passive)
Default Index Profile	D23	0 = Trapezoidal 1 = S-Curve 2 = Parabolic	1.0	0 (Trapezoidal)
Default Jog Profile	D24	0 = Trapezoidal 1 = S-Curve 2 = Parabolic	1.0	0 (Trapezoidal)
Default Maximum Accel	D20	Position Units / Sec^2	D50.D51	0
Default Maximum Decel	D21	Position Units / Sec^2	D50.D51	0
Default Maximum Speed	D19	Position Units / Sec.	D48.D49	0
DH-485 Baud Rate*	D109	4 = 9600 Baud 5 = 19,200 Baud	1.0	4 (9600 Baud)
DH-485 Max Node Address*	D107	$1 \leq D107 \leq 31$	2.0	1

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
DH-485 Node Address*	D106	1≤ D106 ≤ 31	2.0	31
DH-485 Token Hold Factor*	D108	1 ≤ D108 ≤ 4	1.0	4
Drive Fault Action	D44	Same as D40	1.0	0 (Kill Drive)
Droop 1394	D69	RPM	3.n	0
DSP Phase Compensation 1394	D65	0 ≤ D65 ≤ 250 degrees	3.12	151
Dual Loop Axis	D22	Axis Number	1.0	0 (Axis 0)
Encoder Filter Bandwidth	D130	0 ≤ D130 ≤ 1000 Hz	8.4	1000
Encoder Filter Lag Limit	D131	0 ≤ D131 ≤ 1000 Position Units	8.4	0
Encoder Loss Action	D43	Same as D40	1.0	0 (Kill Drive)
Encoder Mode	D8	0 = 4X Quad 1 = Step / Dir. 2 = Count Up / Dn	1.0	0 (4X Quadrature)
Encoder Noise Action	D45	Same as D40	1.0	2 (Status Only)
Fast Home Speed	D17	Position Units / Sec.	D48.D49	0
FGain (Feed Forward Gain)	D2	Unitless	8.4	0
Firmware Version*	D5	ASCII Text	2.0	Current Firmware
Hardware Configured Axes*	D54	2, 4 (Read Only)	1.0	4 2 or 4 (hardware dependent)
Hardware Overtravel Action	D41	Same as D40	1.0	0 (Kill Drive)
Home Position	D10	Position Units	D29.D30	0

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Home Return Speed	D18	Position Units / Sec.	D48.D49	0
IGain (Integral Gain)	D1	Specific to each controller type	7.5	0
K Constant (Conversion Constant)	D9	Counts/Position Unit	15.12	4000 (Counts/ Position Unit) (8192 for the 1394)
Lead Lag Degrees 1394	D81	Degrees	2.n	0
Lead Lag Frequency 1394	D82	Hz	3.n	0
Low Pass Bandwidth 1394	D80	Hz	4.n	0
Maximum Test Safe Speed	D53	Position Units / Sec.	D48.D49	0
Maximum Test Travel	D52	Position Units	D29.D30	0 (10 for the 1394)

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Motor ID 1394	D90	0 = Custom 1 = B410G 2 = B420E 3 = B430E 4 = B515E 5 = B520E 6 = B530E 7 = B410J 8 = B420H 9 = B430G 10 = B515G 11 = B520F 12 = B310H 13 = B330H 14 = B420G 15 = B440G 16 = B460F 17 = B630F 18 = B660E 19 = B690E 20 = B840E 21 = B860C 22 = B720E 23 = B730E 24 = B740C 25 = B220H 26 = B360Z 27 = B660F 28 = B740E	1.0	0
Motor ID End RPM 1394	D95	RPM	4.n	0
Motor ID Slope 1394	D96	% / KRPM	3.n	~0
Motor ID Start RPM 1394	D94	RPM	4.n	0
Motor Inertia 1394	D91	mSec / KRPM	3.n	~0
Motor Poles 1394	D111	$2 \leq D111 \leq 12$ even only		4
Motor Rated Current 1394	D93	Amps	2.n	0

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Motor Rated Speed 1394	D92	RPM	4.n	0
Motor Resolver Offset 1394	D97	Revs	0.2	0
Motor RPM 100% 1394	D104	RPM		0
Motor RPM 300% 1394	D105	RPM		0
MV Turns Range	D115	$1 \leq D115 \leq 256$	3.0	256
Negative Travel Limit	D12	Position Units	D29.D30	-100
NFF Gain 1394	D68	% of Velocity Cmd	3.n	0
Operator Interface Channel*	D38	1 = Serial Port A 2 = Serial Port B	1.0	2 (Serial Port B)
Output Limit	D6	Volts	5.3	10 (Volts) (0 for the1394)
Output Offset	D7	Volts	5.4	0 (Volts)
PCam End Point	D28	+ Integer < 2000 13000 (V3.0 and later)	4.0	0
PCam Start Point	D27	+ Integer < D28	4.0	0
PGain (Proportional Gain)	D0	Specific to each controller type	7.5	0
Position Error Fault Action	D42	Same as D40	1.0	0 (Kill Drive)
Position Error Tolerance	D15	Position Units	D29.D30	0
Position Lock Tolerance	D16	Position Units	D29.D30	0.01
Position Units Decimal Digits	D30	$0 \leq D30 \leq D29$	2.0	3
Position Units Total Digits (Field Length)	D29	$1 \leq D29 \leq 10$	2.0	6
Positive Travel Limit	D11	Position Units	D29.D30	100

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Resolver Loss Action 1394	D98	0 = KILL_DRIVE 1 = STOP_MOTION 2 = STATUS_ONLY	1.0	2
Resolver Poles 1394	D112	$2 \leq$ D112 ≤ 12 even only		4
RIO Adapter Rack Address*	D55	$0 \leq$ D55 ≤ 31	2.0	0
RIO Adapter Rack Size*	D56	0 = ¼ 1 = ½ 2 = ¾ 3 = Full	1.0	0 (¼)
RIO Adapter Starting Group*	D59	0 = 0 1 = 2 2 = 4 3 = 6	1.0	0
RIO Baud Rate	D58	0 = 57.6k 1 = 115.2 2 = 230.4	1.0	0 (57.6k)
RIO Channel*	D57	0 = A 1 = B	1.0	0 (Channel A)
RIO Scanner Delay Time*	D60	$0.2 \leq$ D60 ≤ 5.2 seconds	3.2	0.2 (Seconds)
RIO Scanner Rack Address* (Only for S Class Integrated & Basic)	D65	$0 \leq$ D65 ≤ 7	1.0	0
Self Tuning Increment	D34	Position Units	D29.D30	0.25
Servo Update Rate*	D47	$250 \leq$ D47 ≤ 2000 (Hz)	4.0	1000 (Hertz) 500 (Hertz 500 (Hertz)
Shunt Resistor ID 1394	D137	0, 1, 2, 3, 4, or 5	1.0	0
Shunt Resistor Long Time Constant 1394	D135	10 - 2550 Seconds	3.0	120

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
Shunt Resistor Power 1394	D133	100 - 22000 Watts	5.0	300
Shunt Resistor Short Time Constant 1394	D134	0.25 - 2.55 Seconds	2.2	0.25
Shunt Resistor Value 1394	D132	4- 255 Ohms	3.0	**4**
Shunt STC Weighting Factor 1394	D136	1 - 100%	3.n	5
SLC I/O Update Rate	D117	0 = Slow 1 = Medium 2 = Fast	1.0	0 (Slow)
SLC Number of Input Bits	D118	0 = 8 (0 - 7) 1 = 24 (0 - 23) 2 = 40 (0 - 39)	1.0	0
SLC Number of Input Floats	D119	+ Integer < 16	2.0	0
SLC Number of MO: Floats Integers	 D123 D122	 + Integer < 257 + Integer < 513	 3.0 3.0	 0 0
SLC Number of MI : Floats Integers	 D125 D124	 + Integer < 257 + Integer < 513	 3.0 3.0	 0 0
SLC Number of Output Bits	D120	0 = 8 (0 - 7) 1 = 24 (0 - 23) 2 = 40 (0 - 39)	1.0	0
SLC Number of Outputs Floats	D121	+ Integer < 16	2.0	0
Software Overtravel Action	D40	0 = Kill Drive 1 = Stop Motion 2 = Status Only	1.0	1 (Stop Motion)
Status Display Channel*	D37	1 = Serial Port A 2 = Serial Port B	1.0	2 (Serial Port B)
Status Display Refresh Time*	D39	Seconds	3.2	0.5 (Seconds)

Data Parameters

Parameter	Parameter Number	Units	Value Format	Default
TCam End Point	D26	+ Integer < 2000 13000 (V 3.0 and later)	4.0	0
TCam Start Point	D25	+ Integer < D26	4.0	0
Transducer Cnts per Mtr Rev 1394	D103	Counts / Rev	5.n	4000 (8192 for the 1394)
Unwind Constant	D13	Counts	8.0	4000 (Counts) (8192 for the 1394)
Unwind Fraction Denominator	D139	1 to 1 billion (Integer Only)	10.0	1
Unwind Fraction Numerator	D138	0 to (D139 -1) (Integer Only)	9.0	0
Unwind Reference	D110	-1,000,000,000 ≤ D11 ≤ 1,000,000,000	9.0	0
Vel CCW Limit 1394	D71	RPM	4.n	0
Vel CW Limit 1394	D70	RPM	4.n	0
Vel I Gain 1394	D67	100% IR/KRPM/Sec	4.n	0
Vel P Gain 1394	D66	100% IR/KRPM	3.n	3
Vel Rate Limit 1394	D72	KRPM / Sec	4.n	2040
Velocity Units Decimal Digits	D49	0 ≤ D49 ≤ D48	2.0	3
Velocity Units Total Digits (Field Length)	D48	1 ≤ D48 ≤ 13	2.0	6
Vernier Offset	D116	Position Units	D29.D30	0 (position units)
VGain (Velocity Gain)	D3	Specific to each controller type	7.2	0

Data Bits Table

The following table lists the Data Bits alphabetically by their descriptive name as displayed in the scroll-down display of the Control Settings window. The Data Bit number follows the description. The Disabled and Enabled columns show the value set when either 0 or 1 is selected. The final column shows the Default values that are set when the iCODE `.I` command (Initialize Control) is executed to initialize the software. Data bits marked with an asterisk (*), are not axis specific.

Data Bits

Description	Bit #	Disabled (0)	Enabled (1)	Default
Auto Program Boot*	B29	No	Yes	0
AxisLink*	B48	No	Yes	0
AxisLink Extended Node Enabled	B60	No	Yes	0
AxisLink Virtual Axis	B51	Disable	Enable	0
AxisLink Virtual Axis Commanded	B50	Actual	Command	0
Backlash Takeup Direction	B13	Negative	Positive	0
Common Bus AC/DC Mode 1394	B64	No	Yes	0
Common Bus Enable 1394	B63	No	Yes	0
CNET Enabled 1394	B65	No	Yes	0
DH-485/CNET Auto Update Enabled	B59	No	Yes	0
DH-485 Enabled*	B53	No	Yes	0
Drive Fault Checking	B17	No	Yes	0

Data Bits

Description	Bit #	Disabled (0)	Enabled (1)	Default
Drive Fault Contacts	B18	Normally Open	Normally Closed	0 (1 for the 1394)
Dual Loop Control	B14	No	Yes	0
Encoder Filter Enable	B27	No	Yes	0
Encoder Loss Checking	B25	No	Yes	0
Encoder Polarity	B10	Normal	Inverted	0 (1 for the 1394)
External Velocity Loop	B8	Torque	Velocity	0 (1 for the 1394)
Half Duplex*	B31	Full	Half	0
Hardware Overtravel Checking	B4	No	Yes	0
Hardware Overtravel Contacts	B5	Normally Open	Normally Closed	0
Home Direction	B3	Positive	Negative	0
Home Switch Contacts	B2	Normally Open	Normally Closed	0
Home to Marker	B0	No	Yes	0
Home to Switch	B1	No	Yes	0
Low Line Voltage 1394	B55	No	Yes	0
Multidrop Mode*	B33	No	Yes	0
Op Interface Use Linefeeds*	B21	No	Yes	1
Op Interface Use Returns	B22	No	Yes	1
PCam Table Forward	B39	Backward	Forward	1
PWM Output	B7	± 10V	PWM	0
Remote Mode Use Linefeeds*	B19	No	Yes	1
Remote Mode Use Returns*	B20	No	Yes	1

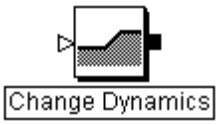
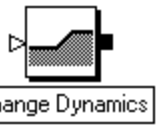
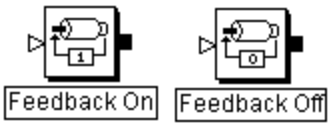
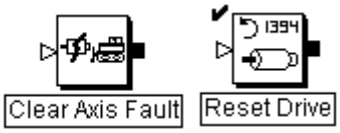
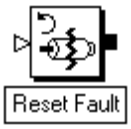
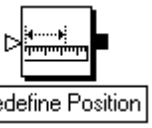
Data Bits				
Description	**Bit #**	**Disabled (0)**	**Enabled (1)**	**Default**
Resolver Loss Checking 1394	B54	Yes	No	0
RIO Enabled*	B44	No	Yes	0
RIO Plug Installed*	B46	No	Yes	0
RIO Scanner Mode*	B45	Adapter	Scanner	0
Rotary Axis	B9	Linear	Rotary	0
Rotary PCams	B38	No	Yes	0
Rotary TCams	B42	Linear	Rotary	0
Servo Output Polarity	B11	Normal	Inverted	0
Shunt Resistor Parameters Enable 1394	B62	No	Yes	0
SLC Interface Enabled	B56	No	Yes	0
SLC M File Enabled	B57	No	Yes	0
SLC M File Transfer Mode	B58	Auto	Manual	0
Software Overtravel Checking	B6	No	Yes	0
Status Display*	B23	No	Yes	1
TCam Table Forward	B43	Backward	Forward	1

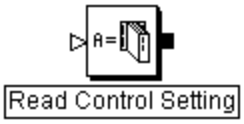
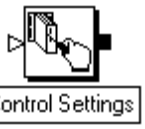
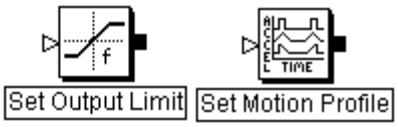
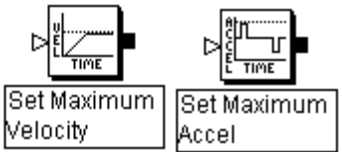
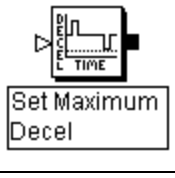
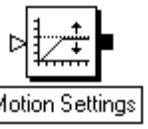
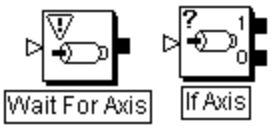
Understanding Differences Between GML 3.x and GML Commander Function Blocks

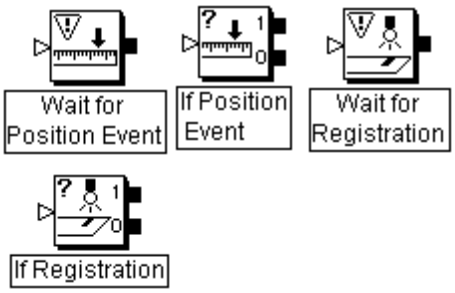
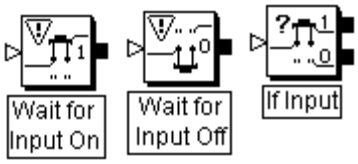
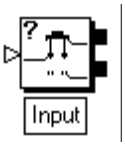
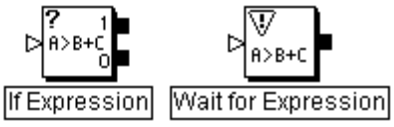
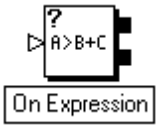
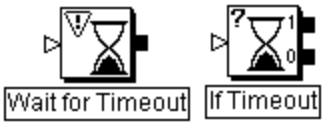
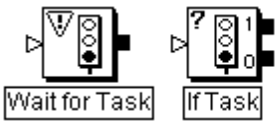
The following tables shows—in the left column—GML version 3.X function block (or blocks), and—in the right column—the corresponding GML Commander function block. These blocks are grouped according to the GML Commander palette, or toolbar, on which the GML Commander function blocks appear.

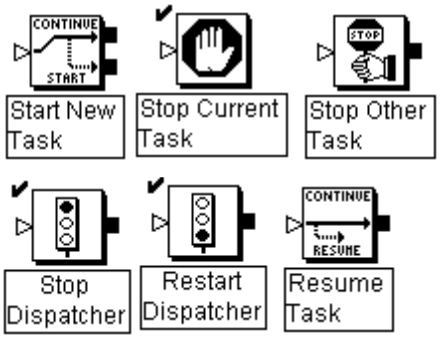
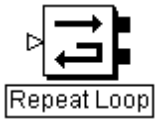
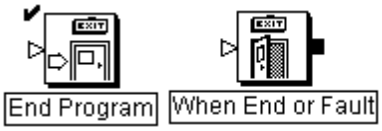
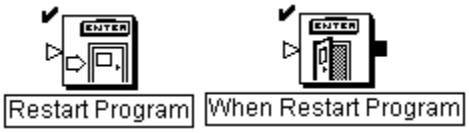
Main Palette Blocks

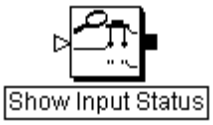
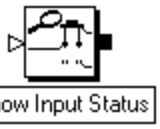
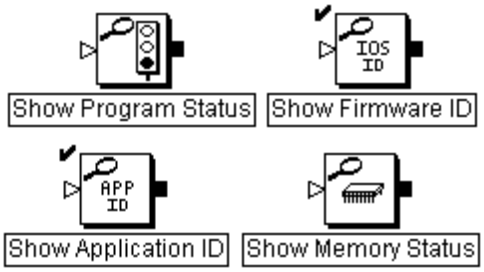
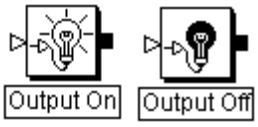
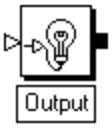
GML version 3.x block(s)	GML Commander block
	
	
	
	

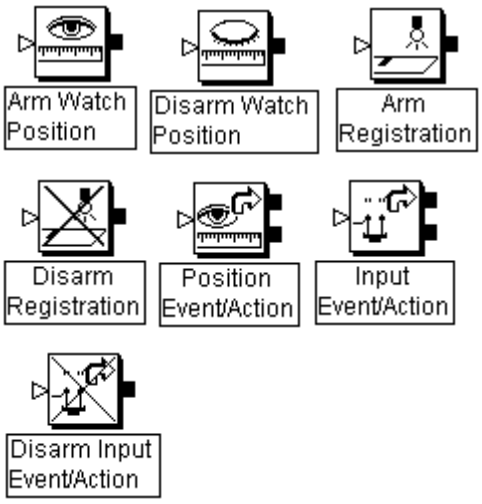
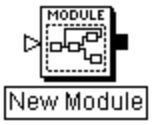
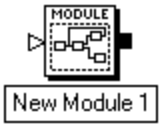
GML version 3.x block(s)	GML Commander block
	
	
	
	
	
	

GML version 3.x block(s)	GML Commander block
  	
  	
	

GML version 3.x block(s)	GML Commander block
	
	
	
	
	
	

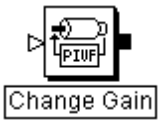
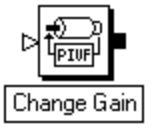
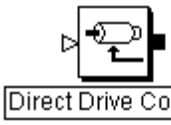
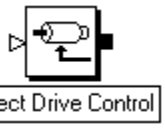
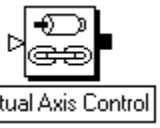
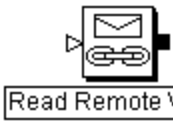
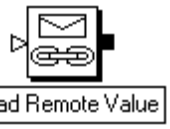
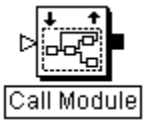
GML version 3.x block(s)	GML Commander block
	
	
	
	
	
	

GML version 3.x block(s)	GML Commander block
	
	
	
	

GML version 3.x block(s)	GML Commander block
	
	
	

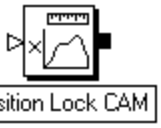
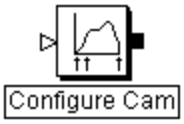
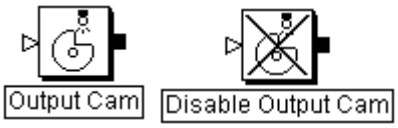
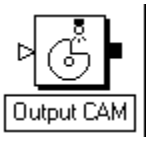
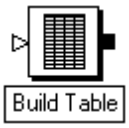
Advanced Palette Blocks

GML version 3.x block(s)	GML Commander block
	

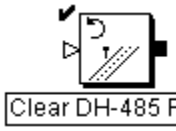
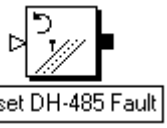
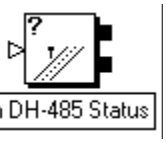
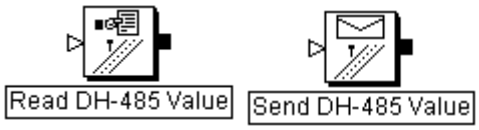
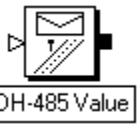
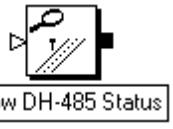
GML version 3.x block(s)	GML Commander block
	
	
	
	
	
	
	

GML version 3.x block(s)	GML Commander block
	

CAM Palette Blocks

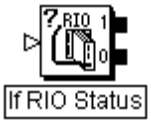
GML version 3.x block(s)	GML Commander block
	
	
	
	
	
	

DH-485 Palette Blocks

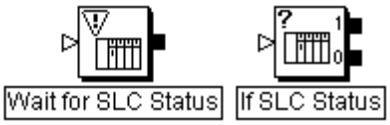
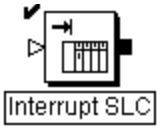
GML version 3.x block(s)	GML Commander block
	
	
	
	

RIO Palette Blocks

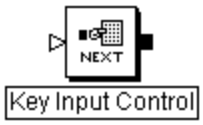
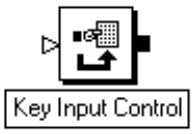
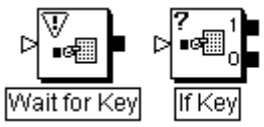
GML version 3.x block(s)	GML Commander block
	

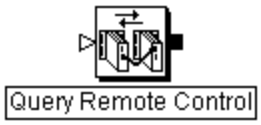
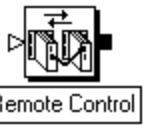
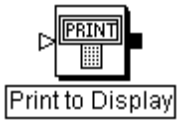
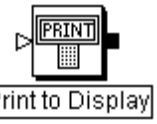
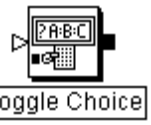
GML version 3.x block(s)	GML Commander block
	
   	
	

SLC Palette Blocks

GML version 3.x block(s)	GML Commander block
	
	
	

RS-232/422 and Multidrop Palette Blocks

GML version 3.x block(s)	GML Commander block
	
	

GML version 3.x block(s)	GML Commander block
	
	
	
	
	
	
	

ASCII Reference

The following tables are provided as a reference for GML Commander blocks that require ASCII code or characters. The character codes, code values, and descriptions are defined by the American National Standards Institute (ANSI) as the American Standard Code for Information Interchange (ASCII). For more information, see ANSI Standard X3.4-1977.

ATTENTION: Although code values are displayed as both decimal and hexadecimal (hex) numbers, always use the decimal value in GML Commander.

ASCII Control Codes

ASCII	Keyboard	Decimal	Hex	Description
NUL	CTRL-@	**0**	**00**	Null
SOH	CTRL-a	**1**	**01**	Start of heading
STX	CTRL-b	**2**	**02**	Start of text
ETX	CTRL-c	**3**	**03**	End of text
EOT	CTRL-d	**4**	**04**	End of transmission
ENQ	CTRL-e	**5**	**05**	Enquiry
ACK	CTRL-f	**6**	**06**	Acknowledge
BEL	CTRL-g	**7**	**07**	Bell
BS	CTRL-h	**8**	**08**	Backspace
HT	CTRL-i	**9**	**09**	Horizontal tab
LF	CTRL-j	**10**	**0A**	Line feed
VT	CTRL-k	**11**	**0B**	Vertical tab
FF	CTRL-l	**12**	**0C**	Form feed
CR	CTRL-m	**13**	**0D**	Carriage return
SO	CTRL-n	**14**	**0E**	Shift out
SI	CTRL-o	**15**	**0F**	Shift in
DLE	CTRL-p	**16**	**10**	Data link escape
DC1	CTRL-q	**17**	**11**	Device control 1

ASCII	Keyboard	Decimal	Hex	Description
DC2	CTRL-r	**18**	**12**	(XON)
DC3	CTRL-s	**19**	**13**	Device control 2
DC4	CTRL-t	**20**	**14**	Device control 3
NAK	CTRL-u	**21**	**15**	(XOFF)
SYN	CTRL-v	**22**	**16**	Device control 4
ETB	CTRL-w	**23**	**17**	Negative acknowledge Synchronous idle
Block				End of transmission
CAN	CTRL-x	**24**	**18**	Cancel
EM	CTRL-y	**25**	**19**	End of medium
SUB	CTRL-z	**26**	**1A**	Substitute
ESC	CTRL-[	**27**	**1B**	Escape
FS	CTRL-\	**28**	**1C**	File separator
GS	CTRL-]	**29**	**1D**	Group separator
RS	CTRL-^	**30**	**1E**	Record separator
US	CTRL-_	**31**	**1F**	Unit separator
DEL		**127**	**7F**	Delete

ASCII Printing Characters

Char.	Dec.	Hex		Char.	Dec.	Hex		Char.	Dec.	Hex
Space	32	20		@	64	40		`	96	60
!	33	21		A	65	41		a	97	61
“	34	22		B	66	42		b	98	62
#	35	23		C	67	43		c	99	63
$	36	24		D	68	44		d	100	64
%	37	25		E	69	45		e	101	65
&	38	26		F	70	46		f	102	66
’	39	27		G	71	47		g	103	67
(	40	28		H	72	48		h	104	68
)	41	29		I	73	49		I	105	69
*	42	2A		J	74	4A		j	106	6A
+	43	2B		K	75	4B		k	107	6B
,	44	2C		L	76	4C		l	108	6C
-	45	2D		M	77	4D		m	109	6D
.	46	2E		N	78	4E		n	110	6E
/	47	2F		O	79	4F		o	111	6F
0	48	30		P	80	50		p	112	70
1	49	31		Q	81	51		q	113	71
2	50	32		R	82	52		r	114	72
3	51	33		S	83	53		s	115	73
4	52	34		T	84	54		t	116	74
5	53	35		U	85	55		u	117	75
6	54	36		V	86	56		v	118	76
7	55	37		W	87	57		w	119	77
8	56	38		X	88	58		x	120	78
9	57	39		Y	89	59		y	121	79
:	58	3A		Z	90	5A		z	122	7A
;	59	3B		[	91	5B		{	123	7B
<	60	3C		\	92	5C		l	124	7C
=	61	3D		]	934	5D		}	125	7D
>	62	3E		^	94	5E		~	126	7E
?	63	3F		_	95	5F				

INDEX

S